1989-03

Implementation of a design for testability strategy using the Genesil silicon compiler

Davidson, John Carl

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A DESIGN FOR TESTABILITY STRATEGY USING THE GENESIL SILICON COMPILER

by

John Carl Davidson

March 1989

Thesis Advisor: Herschel H. Loomis, Jr.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved fpr public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) IMPLEMENTATION OF A DESIGN FOR TESTABILITY STRATEGY USING THE GENESIL SILICON COMPILER

12 PERSONAL AUTHOR(S)
DAVIDSON, John Carl

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 March	123

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Design for Testability; Scanpath; Built-in Test; Genesil shiftable test latch; silicon compiler; linear feedback shift register

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Design for Testability (DFT) is receiving major emphasis in the VLSI design field due to increasing circuit complexity. The utility of the silicon compiler and its value to a system designer without extensive VLSI design experience is discussed. Two major techniques for DFT, Scanpath Design and Built-in Test Design, are implemented using the Genesil silicon compiler. The basic building block, the shiftable test latch, is described in random logic block form and parallel datapath form. Linear feedabck shift registers used as random vector generators and signature analyzers are used in the Built-in Test design. An Automatic Test Generation (ATG) program is used to provide a measure of fault coverage for the two DFT techniques. The appendix is a brief tutorial illustrating the use of the Genesil system's shiftable test latch in its different configurations.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Herschel H. Loomis, Jr.	22b TELEPHONE (Include Area Code) 408-646-3214 22c OFFICE SYMBOL 62Lm

Approved for public release, distribution is unlimited

IMPLEMENTATION OF A DESIGN FOR TESTABILITY STRATEGY USING THE GENESIL SILICON COMPILER

by

John Carl Davidson
Lieutenant, United States Navy
B.A., Iowa State University, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

March 1989

ABSTRACT

Design for Testability (DFT) is receiving major emphasis in the VSLI design field due to increasing circuit complexity. The utility of the silicon compiler and its value to a system designer without extensive VLSI design experience is discussed. Two major techniques for DFT, Scanpath Design and Built-in Test Design, are implemented using the Genesil silicon compiler. The basic building block, the shiftable test latch, is described in random logic block form and parallel datapath form. Linear feedback shift registers used as random vector generators and signature analyzers are used in the Built-in Test design. An Automatic Test Generation (ATG) program is used to provide a measure of fault coverage for the two DFT techniques. The Appendix is a brief tutorial illustrating the use of the Genesil system's shiftable test latch in its different configurations.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to thank my wife, Susan, for her unending support and patience throughout our stay at NPS, I could not have done it otherwise.

I. INTRODUCTION

A. BACKGROUND

VLSI (Very Large Scale Integrated) circuit technology has resulted in the dramatic increase in the circuit density (number of components, gates, circuits or memory bits) contained within a single chip. Along with the increase in the number of circuits has come a corresponding increase in the complexity of circuit testing. VLSI circuits are technical products and it is important for the user to know if the device "works" from a physical standpoint as well as a functional one. There are two questions that arise that provide the impetus for physical testing of a device:

- o Does the device work?
- o Will it continue to work?

These questions determine the availability and the reliability of the chip or system in question.

There is also a third question: "Is it affordable?" that determines the cost effectiveness of the device. The question of cost effectiveness is an important one given the fact that the increasing complexity of VLSI components has resulted in a trend of higher testing costs. It is conceivable that some circuits are so complex that testing them by conventional methods might itself be prohibitive and an otherwise good design might not go into production because there is no way to determine convincingly its reliability.

Testing, in a general sense, means to examine a product to ensure that it functions correctly and exhibits the properties and characteristics that it was designed to possess [Ref. 1:p. 13].

VLSI technology has introduced complexity into the testing of integrated circuits in two important ways. First, the circuits have become so large and complicated that testing cannot be done by an individual. This has made planning and designing for testing more difficult. Computer-aided tools are one solution to this problem. Second, integrated circuits have become so fast and compact as well as being largely inaccessible that new methods of testing are required. Accessibility refers to the ease at which the internal nodes of a device are made available to a testing procedure for control and observation. New methods of testing required to deal with the increase in complexity lead to increases in cost.

Testing consists of supplying a stimuli to the circuit under test and obtaining and comparing its responses to the expected responses. The rapid growth of VLSI circuits has led to a new industry and technology, heavily dependent on the computer to aid in testing. This is the Automatic Test Equipment (ATE) industry. ATE generate test patterns, supply the test patterns to the object under test, obtain the output responses, and compare the response patterns to predicted behavior. Despite the growth of the ATE industry, LSI and

VLSI testing has continued to become more difficult and costly.

Conventional testing, as defined by F. Tsui [Ref. 1:p. 48], is testing that relies primarily on adding improved mechanical means for testing and not on the addition of logic within the design. Design for Testability (DFT), on the other hand, by relying on the addition of logic to facilitate testing, can be considered to be electronic in nature vice mechanical and an integral part of system design.

Conventional testing has three characteristics that differentiate it from DFT:

- o Conventional methods cannot test parts in-system. Component testing must be done in isolation from the rest of the system.
- o Conventional methods rely on test equipment to supply test patterns and capture the output response.
- o Conventional methods require tester-driven timing. The timing control originates from the test equipment and is not considered part of the system timing.

Because of the increasing circuit integration and speeds of VLSI design, conventional test methods have become inadequate. The chief reason is that the methods rely on feeding the signals through some sort of test-interface. With the increasing density of the circuits, more input/output (I/O) pins are required for the normal operation of the chip. However, due to technology constraints, the

miniaturization of the I/O pads has not kept pace with the rate of increasing density within the core of the chip. Thus, the number of I/O pins available for testing has decreased. Also, as the physical size of the chips has decreased, so has the ratio of periphery to surface area resulting in less area available for an increasing number of pins. The use of a test-interface also contributes noise and some signal distortion that might affect the successful implementation of a test.

As mentioned above, although the per-chip fabrication and assembly costs have decreased rapidly as the technology has matured, testing costs have not been reduced. Consequently, as a percentage of the total cost of a product, the cost of testing has continually increased [Ref. 1: p. 15]. Costs of testing include test equipment (hardware), test generation which reflects costs in both test pattern generation and verification, testing time, and testing personnel. The goal of **Design for Testability** (DFT) is to find ways to make testing easier, more efficient, and less costly. It is believed that through the incorporation of testability design from the very beginning of a design project, testing can be made more economical and effective. DFT adds circuits to the object to be designed in order to make it easier to test. These circuits add to the observability and controllability of the system.

Controllability refers to the ease by which a specific signal can be produced at some internal node of a design by applying a signal to the inputs of the design. **Observability** refers to the ease by which the state of an internal node can be determined at the outputs of the design [Ref. 2:p. 100]. These two concepts are important in understanding circuit characteristics that determine testability. This is the chief aim of the work done in this thesis, to demonstrate Design for Testability as it is implemented by the devices available through the Genesil Silicon Compiler, hereafter referred to simply as Genesil.

Testing, at the integrated circuit level, mainly involves combinational logic. Most digital systems are built with mixtures of combinational networks and latches. Latches are difficult to test because they are sequential in nature and the feedback loops inherent in sequential devices are difficult to test. A fault in a sequential circuit would require a sequence of test patterns or vectors to detect it. The method that Genesil uses to handle sequential circuits, time unrolling, will be discussed in Chapter V.

With VLSI circuits becoming more inaccessible, DFT provides ways of gaining access to the interior of the circuit to facilitate testing. The focus of this thesis is to demonstrate the incorporation of additional circuitry within the framework of the design in order to increase testability. **Testability** can be defined as the capability to

examine whether an object is "fault-free". We achieve testability through increasing the controllability and observability of a circuit.

The goal in testing is not necessarily to discover the exact physical failures, often merely detecting the existence of those failures is enough, since it may be that the location of the fault is not necessarily important. In order to detect a fault within a circuit, a sequence of test patters (vectors) is applied to the circuit and the results are compared with those known to belong to a good circuit.

Any difference implies that fault(s) are detected by the test pattern. The total number of faults that can be detected as compared to the total number of possible faults is the fault **coverage**. Physical failures are due to either manufacturing defects or wear-out in the field. Failures occurring during manufacture might include faulty transistors, breaks in lines at some level (polysilicon, metal, diffusion, etc.), and shorts between levels and among levels. Devices in the region of a failure will also be affected. Alignment errors, mask failures, and problems with the lithographic techniques vital to the successful manufacture of a VLSI circuit all contribute to physical failure. They result in pinholes in the oxide, faulty contacts, and defective devices. Improper handling can result in input gate breakdown due to static electricity. Moisture in the packaging of the circuit can lead to failure.

Long term failures result from breaks in lines and shorts between lines. The aluminum metal can start to corrode. High current densities in thin wires can result in metal migration. As the technology ages and existing problems are corrected, new ones will evolve and this further complicates the generation of accurate fault models.

A fault model is used to describe the effect of a physical failure on the performance of the device. A **stuck-at fault** model describes the effect of a physical failure that results in the inputs or outputs of logic gates being permanently stuck at logic 0 or 1. A **bridging fault** model describes shorts between lines at the logic level of the circuit. There are also stuck-open fault models. Many physical failures can be described by the single stuck-at fault model. There are also multiple stuck-at fault models.

Figure 1 shows a simple CMOS inverter constructed of a p-channel transistor and an n-channel transistor. A logic 1 at the input causes the n-channel transistor to conduct bringing the output close to ground or 0. A logic 0 causes the p-channel transistor to conduct bringing the output to be "pulled-up" to a VDD or logic 1. If the inverter is faulty (i.e., has an open line, short between lines, or a failed transistor) what can happen? If the input is shorted to ground (0) then the gate output is permanently at logic 1, the p-channel transistor is always on. The same thing happens if there is a break in the line at A, once any

residual charge has leaked out of the p-channel transistor. If the line is broken at B, the input of a logic 0 will cause the expected output. However, if the input is at logic 1, the p-channel transistor will turn off, but since the line is broken, the n-channel transistor will never turn on and the output will remain at a logic 1 for a period of time dependent on the leakage currents, usually milliseconds. If a constant stream of data is being input to the device, the output will look like a steady logic 1, hence stuck-at-one.

A more complex fault will result if one of the transistors has failed. If the n-channel transistor, for example, were to fail permanently in the logic 1 state, a logic 1 applied to the input would not result in any error. If, however, a logic 0 were applied, both transistors would conduct, leaving the output at some intermediate value between VDD and 0.

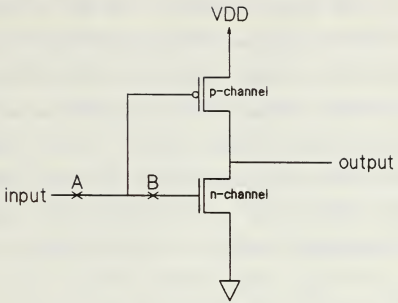


Figure 1. CMOS Inverter Fault Model

Consider the simple NOR gate shown in Figure 2. If there is a break at point C just before the n-channel transistor, the output should normally be logic 0 for A equal to logic 0 and B equal to logic 1. As a result of the failure at C, there is no path for either VSS or VDD to the output. Consequently, the circuit is floating and retains its previous value. The output can be forced to logic 0 by setting A to logic 1 and to logic 1 by setting both A and B to logic 0. The point is that the circuit retains the memory of its previous state and has, therefore, become sequential in nature. This is the stuck-open fault first described by R.L. Wadsak in 1978 [Ref. 3]. The failure would be detected

by forcing the output to a logic 1 (A,B = 0) and then setting B to logic 1; the output would not change if there was a break at C. It is important to note that not all circuits can be described by the fault models described above. Models of functional blocks of logic include shorts between lines in addition to the stuck-at fault models. A short between two lines results in the two lines having the OR or AND of their correct values, depending on the technology used in the device (CMOS technology results in an OR function, NMOS in an AND function). The goal here is not to provide a comprehensive guide to faults but to provide a basic understanding of the effects of some of the physical failures and how they relate to fault models used in testing.

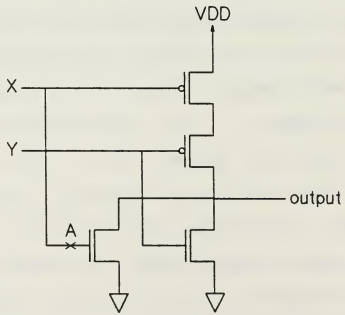


Figure 2. CMOS NOR Gate Fault Model

B. SILICON COMPILATION AND ASIC DESIGN

The device used as the basic test platform in this thesis is a versatile 16-bit correlator. It is a good example of an application-specific integrated circuit, commonly referred to as an ASIC. ASICs have become very popular in military systems due to factors of integration and customization. Avionics systems, for example, require high integration due to size and weight constraints. Other systems, such as those used for communication or targeting, require devices that are high-performance, very specialized, or both. Traditional methods of ASIC design include full-custom design, gate-array circuit design, and standard cell circuit design [Ref. 4:p. 38]. Silicon compilation is the newest method of ASIC design and allows the designer a higher degree of flexibility and feedback than previously available. The silicon compiler works from a high-level description of the circuit that allows the designer to perform successive design iterations quickly and efficiently, providing the designer rapid access to key parameters such as chip size, power consumption, and timing constraints.

The Genesil silicon compiler used at the Naval Postgraduate School in Monterey is particularly effective in that it allows the system designer with little IC design expertise to quickly and effectively create workable circuits. Because of the breadth of the compiler library, including relatively complex circuits such as random access

memory (RAM), read only memory (ROM), programmable logic arrays (PLA), arithmetic logic units (ALU), multipliers, and a host of less complex circuits such as basic logic gates and data-path elements, the designer does not have to design at the transistor level, and in fact, requires little knowledge of this level of VLSI. Figure 3 shows the configuration of the Genesil system at NPS.

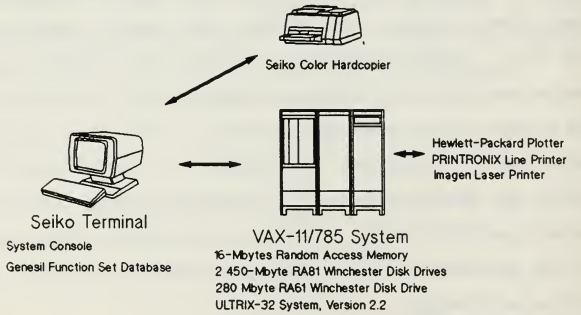


Figure 3. Genesil Silicon Compiler System Configuration at the Naval Postgraduate School

There are two previous theses that describe the use of Genesil [Refs. 5 and 6] and they are highly recommended as background reading for anyone desiring to use the system.

Once the designer has specified his design, the silicon compiler synthesizes its layout. Additionally, simulation models, timing analysis models, and test generation models can be prepared. The compiler, relying on an extensive set of layout rules and circuit design knowledge including information on various fabrication processes, quickly prepares the layout synthesis. The designer can then perform logic simulation to verify the functional performance of the device, timing analysis to determine which paths control the overall system performance, or test generation with the automatic test generation (ATG) module to determine fault coverage. Based on the results of simulation and timing analysis or after examining a list of key parameters resulting from the layout compilation, he can change one or more parameters and quickly examine what effects the changes have on the performance of the system. He can even go so far as to change fabrication techniques.

The silicon compiler is effective because it contains all of the components necessary for circuit design within one tool. Timing analysis is effective because the system "really understands" the circuits it is analyzing. The simulator is effective because each element within the compiler library has been optimized for simulation. This

reduces computation time. Testing analysis is also very efficient at this level. The use of the ATG feature at the compiler level reduces testing time and provides the designer rapid feedback on the degree of controllability and observability available within the design. As this thesis will demonstrate, Genesil has enhanced the basic testability of most designs by making available, within the compiler library, test latches and sequence generators to specifically aid in DFT.

C. THESIS GOALS AND ORGANIZATION

As stated above in several places, the primary goal of this thesis is to demonstrate Design for Testability strategies as implemented by Genesil. Two primary circuits will be demonstrated, the shiftable latch (STL) and the linear feedback shift register (LFSR). Chapter II will describe the original design of the 16-bit correlator chip on Genesil and provide a starting point for the collection of comparison data on simulation, timing and various key parameters such as chip size and power consumption. Chapter III will begin by describing the design of the basic shiftable test latch used in Genesil. The chapter will also detail the latch's incorporation into the basic correlator to enhance testability. Chapter IV will describe the linear feedback shift register used for built-in testing (BIT). It will also detail the use of the LFSR as a random pattern generator and show how it contributes to DFT. Chapter V will

discuss the Automatic Test Generation module and show how it contributes to DFT. Chapter VI will present a summary of the work completed and the conclusions drawn from this research. The advantages of the test latches will be examined, as well as the usefulness of the Genesil silicon compiler in the implementation of the testability strategy. The Appendix will provide a brief tutorial on the use of the testability latches.

II. 16-BIT CORRELATOR TEST CHIP

One of the first research goals was to decide upon a suitable device upon which to implement the various DFT strategies. It was not a requirement that a new circuit be developed; in fact it was desirable to use a chip that had already been designed. It was, therefore, decided to use an integrated chip that had been designed by LT William Galinis, USN and CPT Terence Beck, USA at the Naval Postgraduate School in Monterey, California [Ref. 7]. The chip is an implementation of a versatile low-power CMOS 16-bit correlator. The chip is able to accept data both serially and in parallel and allows the user to specify which bits from an incoming data stream are to be compared to a preloaded reference word. A binary number from 0 to 16 is returned. A 0 represents a perfectly non-correlated signal (anti-correlation) and 16 represents a perfectly matched signal (perfect correlation). Values between 0 and 16 represent a degree of correlation that could be used to decide acceptance or denial of the input data stream. Such a device can be used in many applications, ranging from communications to robotics.

Figure 4 shows the basic correlation equation represented in discrete form, and Figure 5 shows a basic block diagram of the correlation function in digital form [Ref. 8:p. 403-404].

$$\phi_{xy}(mT) = \frac{1}{N} \sum_{k=0}^{N-1} x(kT) \cdot y((k+m)T)$$

Figure 4. Basic Correlation Equation

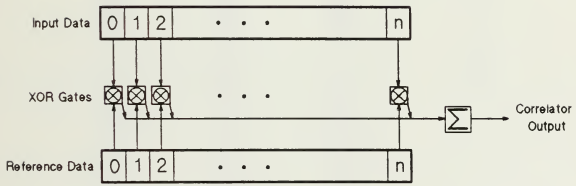


Figure 5. Correlation in Digital Form

Multiplication is implemented by the exclusive-nor (XNOR) function which will yield a 1 if the two bits correlate and a 0 if they do not. The values are then summed and the result is a number between 0 and 16, as explained above.

The correlator circuit used as the test platform is divided into five basic sections: input, xnorreg, combiner, adder, and output. Figure 6 shows the basic block outline of the circuit, as it was designed on Genesil.

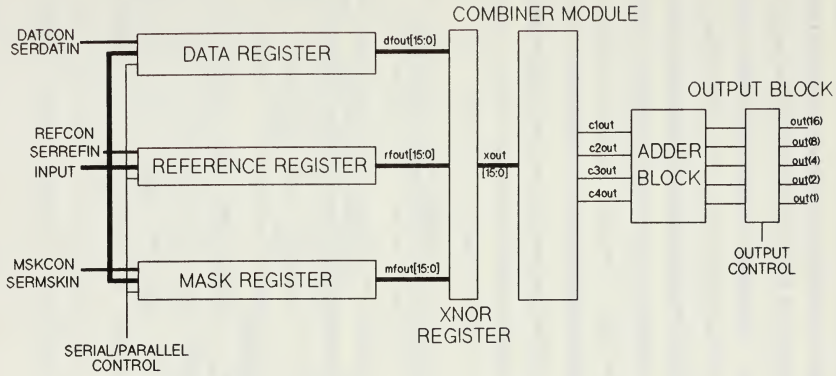


Figure 6. Basic Correlator Chip

The chip was designed in CMOS technology using 1.0, 2.0 and 3.0 micron technology. Micron design rules, commonly used in industry, give a micron resolution of the minimum feature sizes and spacings of the masks required for a given process. In this case, the micron resolution refers to the minimum feature size for polysilicon. The VTC-CP10B fabrication process (fabline) used primarily for the test chip is a VHSIC 1.0μ process (fabline) from VTC Corporation. Other vendors whose fablines are used by Genesil include Honeywell, Motorola, National Semiconductor, and General Electric. The key parameters obtained describing the chip based on several different fablines are shown in Table 1.

TABLE 1
COMPARISON OF FABRICATION TECHNIQUES

FABLINE	VTC-CP10B	AMI-CT20A	GEN-CN30A
AREA (sq.mils)	32695.9	57330.1	111153.3
CORE AREA	17691.5	33987.4	67760.7
AREA PER TRANSISTOR (sq. mils)	8.681864	15.043322	29.166439
POWER DISSIPATION (milliwatts @ 5V @ 10 MHz)	61.51	75.37	100.19

A. BASIC CORRELATOR CHIP DESIGN

1. Input

The input section or module consists of three identical modules: the data module (data_in), the reference module (ref_in), and the mask module (mask_in). Each of the identical modules is a general purpose shift register consisting of 16 D flip-flop/multiplexer combinations. The multiplexers allow the register to be loaded in parallel or serially. The input to each D flip-flop/multiplexer combination is the output of the previous combination (see Figure 7). The signal sp_con will control the multiplexer.

The data register contains the input data to be correlated. The reference register contains the reference word against which the data register is to be correlated. The third shift register is the mask register, it serves as the control for the XNOR register. Placing a "1" in a particular position in the mask register will cause the correlation of the same bit positions in the data and reference registers. A "0" will disable correlation. In this way, flexibility has been added to the device by allowing the user to determine which bits to correlate.

Each register has a simple controller that uses phase_b of the system clock to generate a register clock that operates the D flip-flop. The controller is made up of an AND gate with two inputs, phase_b of the system clock and a

control signal supplied from off the chip. The result of the AND gate is used to clock the D flip-flop on phase_b.

2. XNOR Register

The second section is the XNOR register or xnorreg. It is a random logic block composed of 16 2-input XNOR gates and 16 2-input AND gates. The XNOR register (Figure 5) compares the bits in the data and reference registers. As explained earlier, this corresponds to the multiplication of the two correlation terms. The output of the XNOR register will be a "1" in each bit position where the bits match and a "0" in each position where they do not match or are disabled. The output of the XNOR gate is controlled by the mask register as indicated in Figure 8.

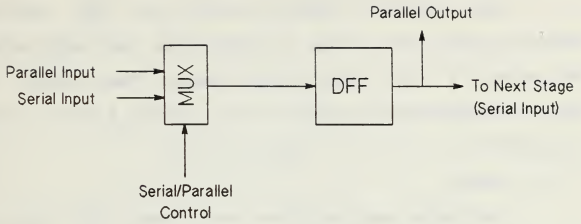


Figure 7. General Purpose Shift Register

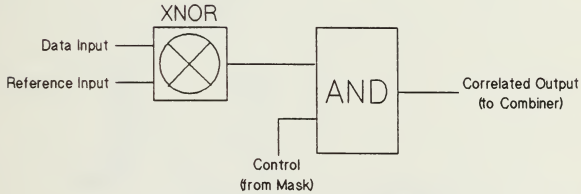


Figure 8. XNOR Register Block

3. Combiner

The combiner module consists of four identical combiner blocks which take 4 inputs from the XNOR register and produce a 3-bit binary-coded decimal (BCD) digit. The logic representation of one of the combiner blocks is shown in Figure 9.

4. Adder

The adder section takes the 4 3-bit BCD digits and adds them together. The result is a 5-bit BCD number with the 5th bit being the carry-out bit of the 2nd stage adder. This block is shown in Figure 10.

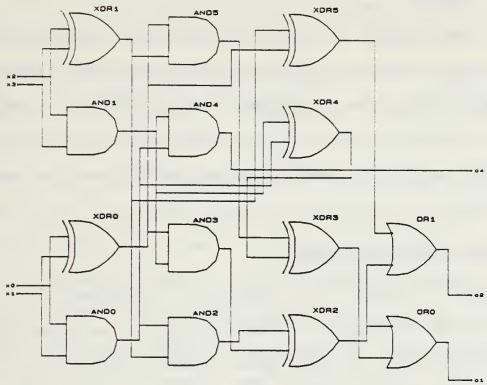


Figure 9. Combiner Block

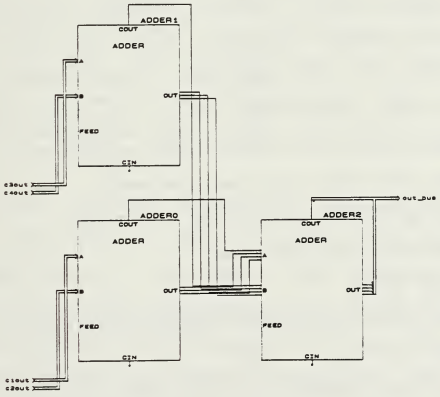


Figure 10. Adder Block

5. Output

The output section of the correlator chip merely consists of a latch circuit made from AND gates. The outputs are made active when a control input, OUTCON, is high. The purpose of this section is to ensure that the output is not available until the input data has settled and is correlated properly.

B. TIMING AND SIMULATION

Genesil provides an efficient environment for timing analysis and simulation. Timing analysis and simulation are run independently of each other thereby increasing the speed of each process. This allows the designer to rapidly evaluate design alternatives.

The Genesil timing analyzer provides timing information based on the physical layout and fabrication technique chosen for the design. After completion of the analysis, a series of reports are produced that provide detailed information on

- Speed at which the object under analysis will run
- Paths that limit the clock frequency
- Duty-cycle (phase high time) constraints
- Input setup and hold times
- Output delays
- Setup and hold times and signal delays for any internal nodes
- Path delays between internal nodes [Ref. 9:p. 1-1].

Genesil uses a two-phase clocking scheme as the timing reference for all clocked devices. It derives the timing characteristics and constraints of the design from switch-level timing models based on the physical design as mentioned above.

Table 2 provides a comparison of the timing information obtained from the Clock Report for each of the various fabrication techniques examined. The Clock Report provides detailed information showing the maximum frequency and the duty cycle limitations of the design that has been analyzed. The timing algorithm reports the symmetric cycle time as the minimum cycle time or twice the longest phase time minus a clock delay calculation, whichever is larger [Ref. 9:p. 4.5]. As indicated in the table, the smaller the fabrication size, the faster the cycle time. Using the VTC-CP10B fabrication technique (in CMOS), the correlator has been analyzed to operate at a maximum of approximately 31.4 MHz. As the fabrication techniques grow larger, the circuit slows to approximately 20.7 MHz.

The Genesil Simulator provides the designer with quick access to the design in order to test design changes or verify functionality. The goal of the simulation is to ensure that the design implementation and the actual layout generated by Genesil work as intended. To achieve this goal, the simulator provides two levels of simulation. The first

TABLE 2
COMPARISON OF TIMING VALUES

FABLINE	VTC-CP10B	AMI-CT20A	GEN-CN30A
PHASE 1 HIGH (ns)	1.9	3.2	6
PHASE 2 HIGH (ns)	15.2	23.6	30.7
MINIMUM CYCLE TIME (ns)	31.8	49.7	65.8
SYMMETRIC CYCLE TIME (ns)	31.8	49.7	65.8

level of simulation is performed with functional models independent of the technology chosen or design layout.

This simulation provides a functional check on the operation of the circuit and is based strictly on circuit design and changes in input signals. This functional simulation uses a **demand-evaluation algorithm.** This algorithm simulates only the minimum amount of logic required to generate a signal value. The user specifies which value is to be checked and then advances time across a clock edge. Requesting a signal value generates the demand that the simulator check that particular net and advancing the clock generates a demand that functional models dependent on the clock edge update and check their internal states. This algorithm runs faster than an event-driven simulation and requires less memory, hence it is particularly suited to functional simulation and iterative checking of design variations.

Once the designer has verified the correct operation of the design, he can move to the next level of simulation. The second level involves the generation of switch-level models that account for the specific technology and layout chosen for the design. The switch-level model is implemented by an event-driven algorithm. This algorithm requires timing information provided by the Genesil timing analyzer. The timing analysis is dependent on process and layout, therefore the switch-level simulation provides an actual simulation of the physical circuit. Signal changes ripple through the design and may change many times before settling into a steady state. Because the signals may change a number of times before settling and many signals are not used at a particular time, the event-driven algorithm uses more memory and is much slower than the demand-evaluation algorithm. If everything is in order, the switch-level simulation should run correctly for the same set of vectors used for the functional simulation. If not, the errors can be traced using special GSLMENU commands and, if necessary, additional test vectors can be created to provide additional initialization. All major sections of the correlator chip were simulated on both levels.

The simulator provides the user with both interactive and batch simulation. Interactive control allows the user to directly stimulate each input and manually advance time. This is ideally suited for verifying functions quickly.

However, it requires that the designer check each output individually to ensure that it is correct. An example of interactive simulation is shown in Figure 11. In this example a value is loaded into the data register by **binding** the values of the input pins, par[15.0], to a binary value of 0111011011101100. This value is then loaded into the data register by advancing the clock one cycle and compared to a value in the reference register, which has already been initialized to a value of 0101010101010101. The output value, seen on cout[4:0], is 01000 which indicates that there were eight matches in the comparison between the value in the data register and the reference register. Manual simulation used in conjunction with the **traceobj** command will generate a test vector file that can be used later to repeat the same sequence of tests.

CORRELATOR_SIMULATION

TIMEPNT	par	dfout	rfout	cout
-1	zzzzzzzzzzzzzzzz	iiiiiiiiiiiiiiii	iiiiiiiiiiiiiiii	iiiii
-1	HHHHHHHHHHHHHHHH	iiiiiiiiiiiiiiii	iiiiiiiiiiiiiiii	iiiii
0	HHHHHHHHHHHHHHHH	iiiiiiiiiiiiiiii	iiiiiiiiiiiiiiii	iiiii
10	HHHHHHHHHHHHHHHH	iiiiiiiiiiiiiiii	iiiiiiiiiiiiiiii	iiiii
10	LHLHLHLHLHLHLHLH	iiiiiiiiiiiiiiii	iiiiiiiiiiiiiiii	iiiii
20	LHLHLHLHLHLHLHLH	iiiiiiiiiiiiiiii	0101010101010101	iiiii
20	LHHHLHHLHHHLHHLL	iiiiiiiiiiiiiiii	0101010101010101	iiiii
30	LHHHLHHLHHHLHHLL	0111011011101100	0101010101010101	01000
40	*LHHHLHHLHHHLHHLL	*0111011011101100	*0101010101010101	*01000

Figure 11. Interactive Simulation of Correlator Chip

Batch simulation uses check functions and test vector files. Test vector files and check functions run faster than the interactive simulation, will generate an error message if the expected result does not agree with the actual result, and provide a standard set of vectors for simulation. Test vectors, in addition to being created with the traceobj command as explained above, can also be written using MASM, a macro-assembler that allows the user to define and use an assembly language customized for the circuit to be simulated [Ref. 10:p. 5-11]. The file can be written in either source code or object code. Examples of each are shown in Figure 12. Figure 12a is the object code written to test the combin0 section of the combiner module. The object code consists of a heading that includes the inputs and outputs of the circuit to be simulated and a data section that lists the input vectors and output vectors. A simulation that produces an output other than that specified in the vector file will generate an error. Figure 12b shows the source code written to simulate the parallel operation of the data_in module. The source code follows a specific format that allows the designer to write the functions that he desires to simulate. An object code file is then generated and the circuit is simulated as described above.

```
CODEFILE
INPUTS xout[3:0];
OUTPUTS clout[2:0];
CODING(ROM)
@5 <0000 >000;
@10 <0001 >001;
@15 <0010 >001;
@20 <0011 >010;
@25 <0100 >001;
@30 <0101 >010;
@35 <0110 >010;
@40 <0111 >011;
@45 <1000 >001;
@50 <1001 >010;
@55 <1010 >010;
@60 <1011 >011;
@65 <1100 >010;
@70 <1101 >011;
@75 <1110 >011;
@80 <1111 >100;
END
```

(a)

```
$define Sig Signal
$define In Input
$define Out Output
$define Pos Position
$define E Expression
$define rep4(a) a a a a

Fields [
    par (Pos = 0, In,Length=16 ) {}
    ser_0 (Pos = 16, In,Length=1, Sticky) {}
    sp_con (Pos = 17, In,Length=1, Sticky) {}
    datcon (Pos = 18, In,Length=1, Sticky) {}

/* Output Fields */

    dfout (Pos = 0, Out,Length=16, Shift = 10) {}
}

Templates [
            serial = sp_con\1;
            parallel = sp_con\0;
            on = datcon\1;
            off = datcon\0;
            load {} = par\@0, dfout\@0;
}

Lineaction:: E(.=.+10), E(temp++);

Data [
/*parallel operation*/
E(temp = 0), parallel, on,
load [temp];
load [temp];
load [temp];
load [temp];
load [temp<<1];
rep4(rep4(rep4(load [temp<<1];)))
rep4(rep4(load [temp<<2];))
]
/*end of source file*/
```

(b)

Figure 12. Simulation Techniques Using Object and Source Code

Check functions are written in a simulation language called GENIE. GENIE (Genesil Interface Extension) is an interpretive language used as a command language in application programs. It has many similarities to C, using similar syntax and many of the same control structures [Ref. 11]. GENIE is an alternative to using test vectors. The test vectors generated by the check functions are captured by the traceobj command. An example of a check function is given in Figure 13. This function was used to verify the operation of the adder block.

```
/* THIS FILE WILL TEST THE ADDER MODULE */
func addtest {
     vars a b c d res
     for a 0 4 {
         for b 0 4 {
             for c 0 4 {
                 for d 0 4 {
                     sn c1out @a
                     sn c2out @b
                     sn c3out @c
                     sn c4out @d
                     set res (+ a b c d)
                     checkatr 10 out @res
                     }
                 }
             }
         }
}
```

Figure 13. Check Function Simulation of Adder Block

The Genesil VLSI implementation of the correlator chip using the 1μ technology resulted in a chip with a total area of 32,695.9 square mils and a total power dissipation of 61.151 milliwatts. This could be compared with the MOSIS 3μ implementation of the correlator chip which had a total area of 48,484.096 square mils. As expected, the Genesil implementation of the chip is smaller, mainly due to the smaller feature size. However, the Genesil implementation of the 2μ feature size is larger than the MOSIS chip, an area of 73,038 square mils compared to 48,484.9 square mils. This demonstrates one of the disadvantages of silicon compilation, the design might not be optimized for size. A full custom tool, such as MAGIC, can generally produce a smaller design if desired. A breakdown of some of the key parameters of the various components of the correlator chip is shown in Table 3. A routing diagram showing the layout of the various modules and blocks that make up the correlator chip is shown in Figure 14.

TABLE 3
KEY PARAMETERS FOR CORRELATOR CHIP

MODULE	TYPE	# OF TRANS.	AREA (sq. mils)	POWER DISSIPATION (mW @ 5V @ 10 MHz
CORLAT_CHIP1	CHIP	3766	32695.9	61.51
ADDER	RANDOM_LOGIC	300	169.1	1.9
CLOCK	PAD	24	582.2	4.8
COMBINER	MODULE	480	439.0	3.44
DATA_IN	MODULE	742	2464.8	6.25
DATAOUT	PAD	90	742.4	21.4
DATCON	PAD	15	148.6	.37
INPUT	PAD	240	2375.5	5.9
MASK_IN	MODULE	742	2496.0	6.25
MSKCON	PAD	15	148.6	.37
OUTCON	PAD	15	148.6	.37
OUTPUT	RANDOM_LOGIC	30	16.5	.26
REFCON	PAD	15	148.6	.37
REF_IN	MODULE	742	2432.0	6.25
SERDATIN	PAD	15	148.6	.37
SERMSKIN	PAD	15	148.6	.37
SERREFIN	PAD	15	148.6	.37
SP_CON	PAD	15	148.6	.37
XNORREG	RANDOM_LOGIC	256	123.3	2.1
VDD	PAD	0	148.6	0
VSS	PAD	0	148.6	0

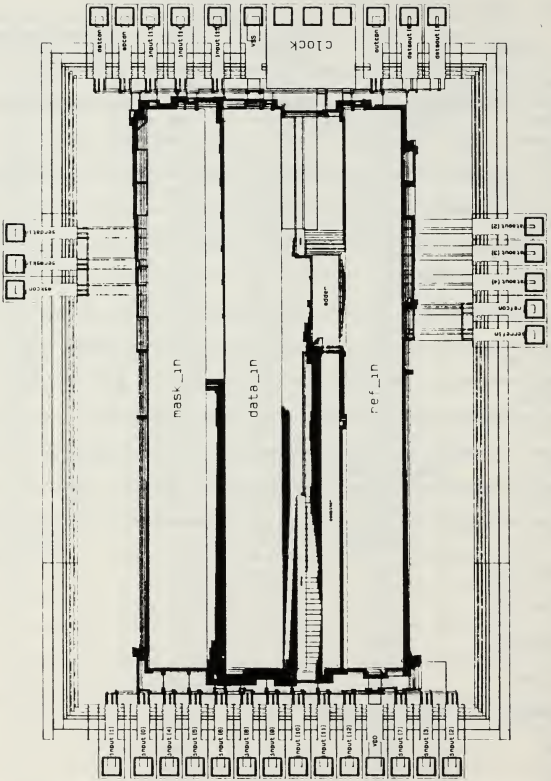


Figure 14. Routing Diagram for Correlator Chip

III. SCANPATH DESIGN FOR TESTABILITY STRATEGY

A. SCANPATH DESIGN

The first Design for Testability strategy investigated was the Scanpath technique. The Scanpath technique strives to enhance the observability and controllability of internal nodes that are inaccessible from the periphery of the system. As mentioned earlier, observability refers to the primary outputs of the design and controllability refers to the ease by which a specific signal can be produced at some internal node by applying a signal to the primary outputs of the design. To increase the observability and controllability scanpaths are added to the design. The scanpaths serve to partition the design into smaller subsystems that are separately more testable than the design as a whole. Figure 15 shows how a generic circuit might be partitioned by a scanpath into individually testable units [Ref 12:p. 374].

The Genesil silicon compiler implements the scanpath using the shiftable test latch in its basic configuration. The STL consists of a data latch in parallel with two serial register latches. The designer builds registers of STLs that are connected via serial inputs and serial outputs. Consequently, the number of peripheral connections are kept at a minimum; only the serial input of the first register and the serial output of the last register are required for vector manipulation.

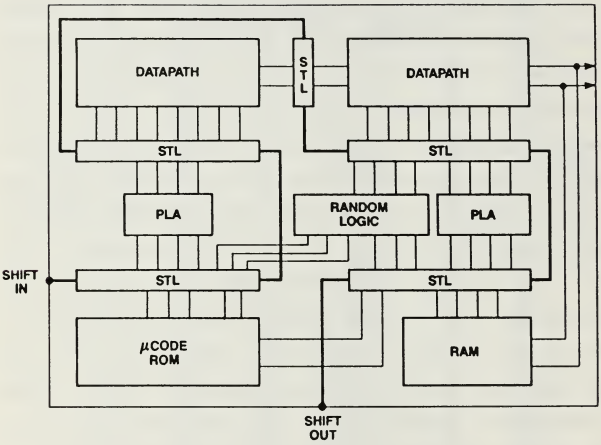


Figure 15. Generic Circuit with Scanpath

The STL enhances the controllability of the design by overriding the contents of the data latch. Internal nodes can be set to arbitrary patterns by shifting a user-defined test vector into the data latch via the serial register. The test vector might be an arbitrary set of bits or a specific vector created by an automatic test vector generation

program. In this thesis, the Genesil Automatic Test Generation (ATG) program was used to generate and evaluate test vectors and fault coverage. The ATG program is discussed in Chapter 5. The STL enhances observability by capturing the states of the internal nodes and shifting the resulting patterns out of the circuits via the serial register.

B. THE SHIFTABLE TEST LATCH

The basic STL consists of three latches and five control gates as shown in Figure 16. The latch labeled D forms the data latch and the latches labeled A, B, F, S, and LOAD govern the flow of data between the data latch and the serial register. To build a test register, the STL's are cascaded. The TOUT connection of each STL is routed to the TIN connection of the next-most-significant STL. By connecting strings of registers together and combining the control signals for each separate test register, a large number of test latches can be used with a minimum overhead in additional pad requirements and external circuitry. Only the TIN of the first STL, the TOUT of the last STL, and the control signals require pads.

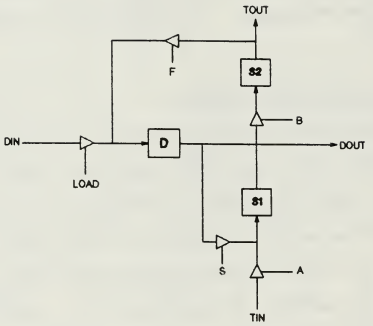


Figure 16. Shiftable Test Latch

The basic STL performs different functions in response to the control inputs. During normal operation, the data latch (Latch D) serves as a storage element and the LOAD signal is driven by phase_b of the systems clock. The **force** operation requires that the test vector be shifted serially into the shift register section (Latches S1 and S2). It is then loaded in parallel into the data latch section and applied to

the circuit. The **sample** operation samples the state of the data latch. Its contents are loaded in parallel into the shift register and then shifted serially out of the circuit. The **shift** function shifts data in the shift register one bit position. There is also a **swap** function. The swap function allows data to be exchanged between the data latch and the shift register. This allows a test vector to be shifted into a circuit as sampled data is shifted out [Ref. 13:pp. 15.2-15.5].

There are three different implementations of the STL when it is created as random logic block. Random logic blocks are used for simple small-scale logic functions. In the unclocked implementation, the control signals are driven by external strobes that are usually generated off-chip. As a result, the designer must ensure that the control signals are generated in the correct sequence and are properly timed. The data latch is driven by a two-phase system clock. The second implementation is the globally clocked model. The control signals mentioned above are combined by additional logic within the block into two signals named M1 and M2. These two signals produce properly timed sequences of the control signals A, B, F, and S. The timing is defined relative to the system clock used to load the data latch. The final implementation of the STL is the locally clocked implementation. This model uses a local two-phase clock that is independent of the system clock. The control signals M1

and M2 are defined relative to the local clock and are derived as mentioned above. The local clock might be generated off-chip or by additional circuitry on the chip.

The testability latches can also be configured using the parallel datapath block available in Genesil. Parallel datapaths are blocks that are specifically tailored for parallel data and control operations such as arithmetic functions and register-file address generation [Ref. 14:p. 1.1]. The same implementations are available in the parallel datapath as were available in the random logic blocks discussed above. There are two additional DFT configurations, generator and signature analyzer, available as parallel datapaths. They will be discussed in Chapter IV.

Table 4 shows a comparison of area, number of transistors, and power dissipation for a 16-bit testability register using each method of implementation. As can be seen from the table, the clocked modes of operation are larger due to the addition of the control circuitry. Also note that the parallel datapath implementations are much larger than the random logic implementations of the testability registers. Parallel datapath blocks require the addition of special interface blocks on input and output. Figure 17 shows the VLSI layout of a 16-bit globally clocked test latch. The control section is on the left side of the layout and the 16 STLs are designed to fit together side by side. Contrast this to Figure 18 which shows the layout of a 16-bit globally

clocked test latch implemented as a parallel datapath. The blocks in the datapath are arranged in a horizontal row with each block of equal height. The height is determined by the width of the datapath which, in this case, is 16. One of the drawbacks of silicon compilation as a design methodology is that the designer has no control over the layout at a level lower than the block level. Once the designer has specified the object type and completed the necessary forms, the layout of the block is done automatically.

TABLE 4
16-BIT TESTABILITY REGISTER COMPARISON

IMPLEMENTATION	AREA (sq. mils)	# OF TRANSISTORS	POWER DISSIPATION (mW @ 5V @ 10MHz)
UNCLOCKED RANDOM LOGIC	245.8	449	3.3
LOCALLY CLOCKED RANDOM LOGIC	257.5	511	3.2
GLOBALLY CLOCKED RANDOM LOGIC	257.5	511	3.2
UNCLOCKED PARALLEL DATAPATH	425.9	617	4.2
LOCALLY CLOCKED PARALLEL DATAPATH	429.6	679	4.4
GLOBALLY CLOCKED PARALLEL DATAPATH	429.6	679	4.4

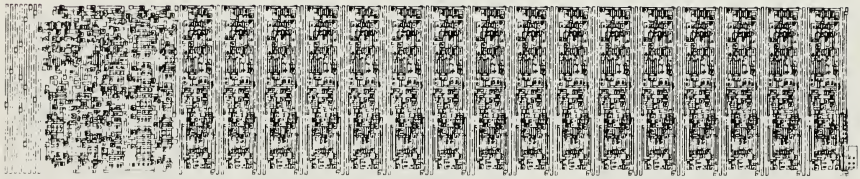


Figure 17. 16-Bit Random Test Register

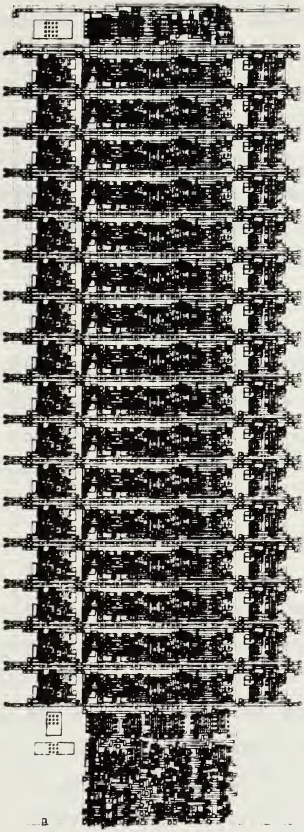


Figure 18. 16-Bit Parallel Datapath Test Register

C. IMPLEMENTING SCANPATH DFT INTO THE CORRELATOR CHIP

The initial DFT approach using Scanpaths was to build a 16-bit register out of STLs and break up the datapath in the correlator chip by putting one or more registers within the data flow. It was quickly decided that the chip was not complex enough to require more than one such register. The problem then became where to place the testability register in order to most increase the observability and controllability of the design.

In order to develop some basis for deciding where to place the register, the fault coverage of the correlator without DFT was determined. The ATG program evaluates the testability of a design and generates a specific set of test vectors designed to provide the optimum amount of fault coverage. Once the set of vectors has been determined, the set may be saved and after the design has been returned from the manufacturer, the test vectors can be used to detect any physical flaws.

The initial fault coverage results for the basic correlator design without any DFT is shown in Table 5. The fault coverage was 92.47% for the entire circuit. The $dummy block is a dummy module used by the ATG program to contain any artificial constructs it creates to carry out the fault evaluation. In order to achieve the specified fault coverage, 605 test vectors were generated. This took a total CPU time of 14 minutes and 28 seconds. It was discovered

that continuing runs of longer time periods did not yield higher coverage. The ATG program will run for a specified amount of time or until achieving a specified degree of fault coverage. There are some internal limits in the program that will cause it to terminate testing but, generally, programs can run for many hours. This means that in order for a fault to be detected, assuming it is one of the faults covered, the set of test vectors will have to be applied, one at a time, to the circuit. If that fault exists, it will be detected when the output vectors are compared to the known correct test results. If one or more of the output vectors do not agree, the fault is detected. The location of the fault is not determined, only its existence.

TABLE 5
ATG FAULT COVERAGE RESULTS FOR BASIC CORRELATOR DESIGN

DEVICE	MODULE	VECTORS	TESTS	FAULT COVERAGE %
BASIC CORRELATOR		605	1215 of 1314	92.47
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		48 of 48	100

Based on the initial set of results, the test register was placed between the xnorreg and the combiner module as shown in Figure 19. It is important to point out that the only analysis done was to look at the information given in Table 5 and determine that the combiner/adder section of the design seemed to be the least testable and a register placed at the input to this section of the design might help to increase the testability. As can be seen in the test results shown in Table 6, the inclusion of the register in this location did not increase the testability of the design. It did, however, reduce the number of test vectors required to get the same amount of coverage. As a result, a more detailed analysis was done using the ATG program.

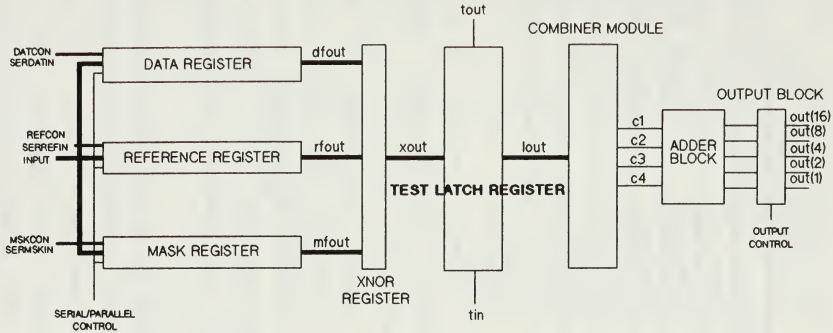


Figure 19. Scan Path Design #1

TABLE 6
THE ATG FAULT COVERAGE RESULTS FOR SCAN DESIGN #1

DEVICE	MODULE	VECTORS	TESTS	FAULT COVERAGE %
SCAN DESIGN #1		207	1255 of 1354	92.68
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		88 of 88	100

The ATG program, via the ANALYZE command, provides specific information about the areas of the circuit that were or were not tested. The circuits can be determined hierarchically to determine which blocks had good and poor coverage and to determine which specific tests were or were not **instantiated** [Ref. 15:p. 5.2]. A fault is said to be instantiated if it can be observed at the primary outputs of the circuit.

Figure 20 shows one of the combiner blocks discussed in Chapter 2. By using the ANALYZE function after a new set of test vectors was generated reflecting the addition of the testability register, it was discovered that a certain test pattern, in this case a logic 1 applied at both inputs, was

not instantiated at the AND gate labeled AND5. The reason is that the test vector required at the primary inputs to produce the test pattern at the AND gate also produced a logic 1 input to the OR gate labeled OR0. The logic 1 overrides the signal coming from the AND gate. The result is that the specific AND test is never instantiated and the fault cannot be determined. The nodes are controllable and can be set to the required pattern but the desired result, a logic 0 for a fault-free circuit and a logic 1 for a faulty circuit, cannot be observed at the primary outputs due to the overriding logic 1 at the OR gate.

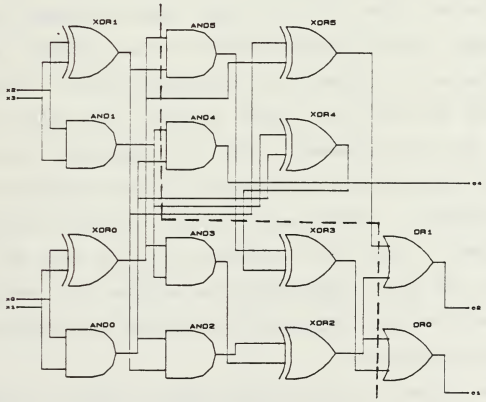


Figure 20. Combiner Block

In order to make the combiner module more observable, it was redesigned so that the OR gate mentioned above could be made more controllable. The combiner module was split into two parts. The split is shown by the dotted line in Figure 20. The new DFT design is shown in Figure 21 and the fault coverage results are shown in Table 7. The table compares the fault coverage for each of the scan designs mentioned and the basic correlator chip without any DFT. Note that although the overall fault coverage only increased by 2.6%, the adder and combiner modules have increased by 5.88% and 16.67%, respectively. The design is fairly small, so the initial fault coverage is expected to be high. The significant point here is the large increase in fault coverage in the adder/combiner area brought about by redesigning the circuit to incorporate DFT. Without the addition of the test vector register, the fault coverage could not be increased. This is important for critical designs where failure in the field, if not detected, could be catastrophic. Additionally, only 133 test vectors were generated. This is a reduction of 78% from the design without any DFT. Extrapolate the savings into reduced test time and increased efficiency of testing and the use of DFT becomes significant.

The tradeoff in the above design is that a 32-bit test register is now required. The STL can only be expanded to 16 bits in random logic so two 16-bit registers have to be

combined or a single 32-bit register can be constructed using the parallel datapath block. Some of the key parameters for these options are shown in Table 8.

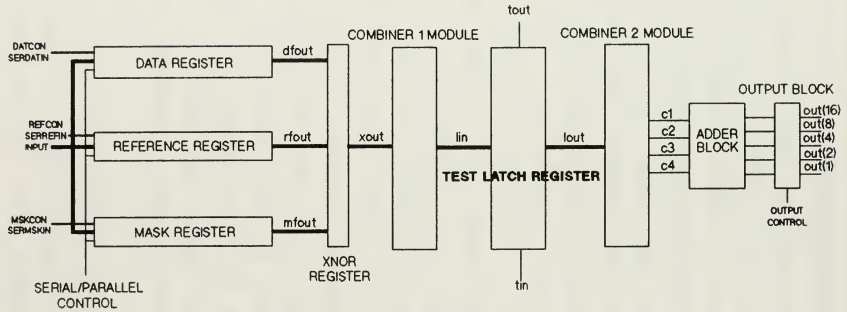


Figure 21. Scan Path Design #2

TABLE 7
FAULT COVERAGE COMPARISON FOR SCAN DESIGNS

DEVICE	MODULE	VECTORS	TESTS	FAULT COVERAGE %
BASIC CORRELATOR		605	1215 of 1314	92.47
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		48 of 48	100
SCAN DESIGN #1		207	1255 of 1354	92.68
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		88 of 88	100

TABLE 7 (cont.)

SCAN DESIGN #2		133	1310 of 1378	95.07
	ADDER		164 of 170	96.47
	COMBINER 1		112 of 112	100
	COMBINER 2		80 OF 80	100
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		101 OF 112	90.18

TABLE 8
32-BIT REGISTER COMPARISON

	AREA (sq. mils)	# OF TRANSACTIONS	POWER DISSIPATION (mW @ 5V @ 10MHz)
PARALLEL DATAPATH	769.9	1271	8.1
RANDOM LOGIC	569.3	1068	6.3

D. STL AND CLOCKING OPTIONS

Genesil uses two-phase non-overlapping clocks derived from a single system clock as the reference for all clocked devices that are logically associated with that system clock [Ref. 9:p. 2.1]. The STL, as described above, can be either clocked by the global system clock or the serial register latches can be clocked by a separate local clock. Separating the clocks makes it possible to halt the normal operation of the chip and scan out the current values of the nodes covered by the STLs. The use of a local clock also allows test vectors to be scanned in more rapidly, thereby reducing time lost to testing. Table 9 shows the difference in clock times for globally and locally clocked DFT designs. Locally clocked STLs provide better performance but will require additional logic circuitry to produce the two test clock phases. If the clock signals are produced on-chip, a single additional input must be provided and the two phases can be produced with a clock processor block. The control signals are identical for both globally and locally clocked STLs.

TABLE 9
COMPARISON OF CYCLE TIMES FOR STL IMPLEMENTATION
OF 16-BIT TEST REGISTER

DEVICE	CLOCK	MINIMUM CYCLE TIME	SYMMETRIC CYCLE TIME
BASIC CORRELATOR	SYSTEM	11.1 ns	16.3 ns
WITH GLOBALLY CLOCKED STL	SYSTEM	23.0 ns	23.0 ns
WITH LOCALLY	SYSTEM	16.1 ns	16.3 ns
CLOCKED STL	LOCAL	12.1 ns	12.2 ns

IV. BUILT-IN TEST DESIGN FOR TESTABILITY STRATEGY

A. BUILT-IN TEST DESIGN

The second Design for Testability strategy investigated was the Built-in Test (BIT) technique. The Scanpath technique discussed in the previous chapter was aimed at enhancing the controllability and observability of the internal nodes of the circuit. The test vectors used to check the circuit are generated by a test vector generation program and then applied and checked by separate test equipment. As the designer requires more detailed testing and the circuit to be tested become more complex, the cost of testing increases. Some other problems associated with using the Scanpath DFT technique include the amount of time required to generate the set of vectors, the size of the set of test vectors becoming too large to be easily handled by the test equipment, and the time taken to apply each vector [Ref. 16:p. 21].

Built-in test techniques attempt to facilitate testing by moving some or all of the test functions onto the chip. The test vectors are generated and can also be analyzed by special circuitry included as part of the functional design. The devices used in this part of the research include a linear feedback shift register (LFSR) to generate the test vectors and a signature analyzer to evaluate the response. The BIT technique has some of the same drawbacks as does the

Scanpath technique -- they both take up some additional area and add some path delay. The compensation is that an outside tester is no longer required and testing time is reduced. Another advantage to Built-in Test is that the circuit can be tested at speed of normal operation and while the circuit is in normal use.

The linear feedback shift register and signature analyzer are implemented as parallel datapath modules by the Genesil silicon compiler. The testability registers can be configured as LFSRs or signature analyzers or both. The STL signature configuration uses the same circuitry and does the same work as the LFSR configuration. It also contains additional logic that allows the designer to combine the sequence of values received from other blocks within the same dataflow and create a **signature** that is unique to that sequence of values. If the signature does not agree with a correct value (determined beforehand via simulation) a fault has been detected. Figure 22a and 22b show the layout of a 16-bit generator and signature parallel datapath. Note the added dimensions of the control section, located at the bottom of the module, of the signature block. This is due to the added circuitry. Also it is easy to determine the vertical nature of the parallel datapath layout. The control section is at the bottom and top, the datapath consisting of the 16 STLs is located in the center, and the interfaces are arranged on the right and left sides of the module. Table 10

shows a comparison of 16-bit test register implementation using generator and signature configurations.

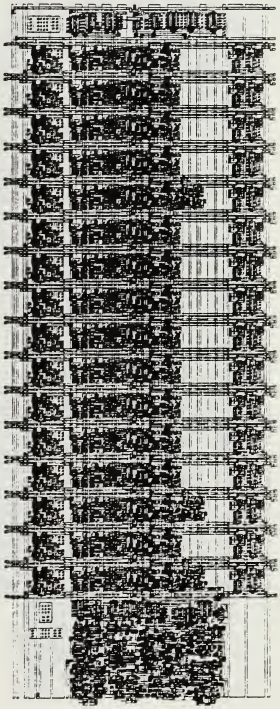


Figure 22a. 16-Bit Generator Layout

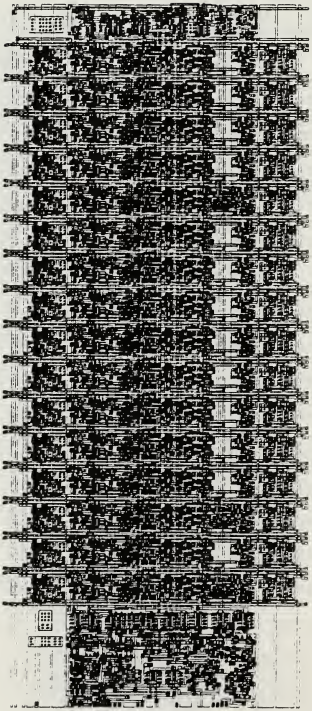


Figure 22b. 16-Bit Signature Layout

TABLE 10
16-BIT TEST REGISTER COMPARISON OF GENERATOR AND ANALYZER CONFIGURATION

IMPLEMENTATION	AREA (sq. mils)	# OF TRANSISTORS	POWER DISSIPATION (mW @ 5V @ 10 MHz)
VECTOR GENERATOR	470.0	679	4.3
SIGNATURE ANALYZER	509.4	835	43.6

Figure 23 shows the general concept behind the DFT strategy implementing Built-in Test. In the first part of the test the STL register on the left acts as a LFSR and generates a string of vectors that become tests for logic block 1. The results of the tests are fed through the second STL register which is configured as a signature analyzer. After a certain number of tests have been generated, the signature can be shifted out of the STL register and checked for any faults. Then, the STLs can be reconfigured via the control inputs so that their configurations are reversed. In the second part of the test, the test patterns are generated by the second STL register, fed through logic block 2 and the resulting signature is produced by the first STL register. [Ref. 17:pp. 392-393]

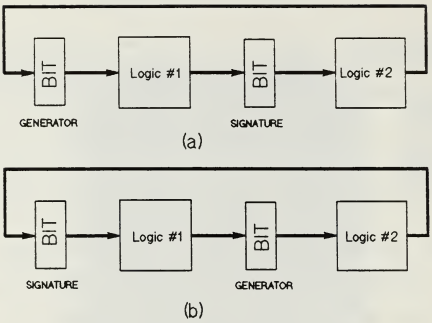


Figure 23. General DFT Built-in Test Strategy

B. LINEAR FEEDBACK SHIFT REGISTERS

The set of test vectors produced by the LFSR, as implemented by Genesil, is a set of pseudorandom vectors. They are called pseudorandom because the vector set is produced by a known circuit, however, the set exhibits many properties of random signals. These characteristics are given a detailed discussion in Golumb [Ref. 18]. The LFSR consists of a series of delay elements such as flip-flops with no external inputs and feedback paths through XOR gates as shown in Figure 24. The R input determines whether data is shifted into the lowest significant bit from the serial input of the XOR feedback path.

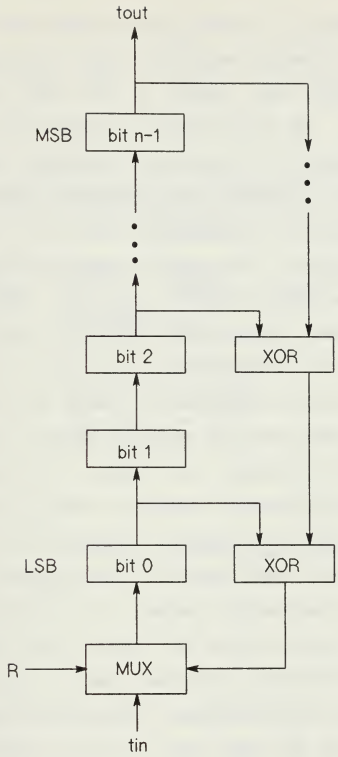


Figure 24. Genesil LFSR Configuration

The arrangement of the XOR gates within the feedback is determined by a constant called the LFSR polynomial. This constant determines the length of the LFSR pseudorandom vector sequence. For most testing applications, the set of test vectors should be as long as possible. The longest sequence, called a maximal length sequence, results if the constant determining the feedback chain is a primitive polynomial. In general, the maximal length sequence of an n-bit LFSR contains $2^n - 1$ vectors [Ref. 2:p. 134]. Peterson [Ref. 19] provides extensive coverage of linear feedback shift registers and vector sequences. Genesil provides the designer with a default polynomial to meet the criteria for maximal length sequences for datapath widths from 4 to 34 bits wide. The designer can change the value of the polynomial if an application requires a specific polynomial.

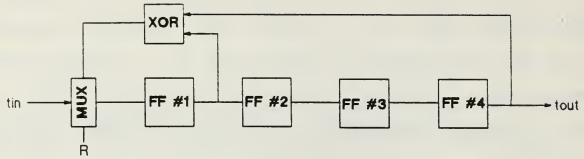
Figure 25 shows the Genesil form for the generator function of a 4-bit test register. The polynomial constant is entered as a hexadecimal number. In the LFSR the most significant bit always starts the chain of XOR gates so the highest order coefficient is always one. The lowest bit always feeds into the multiplexer controlled by the R input and is also always one [Ref. 14:p. 24.14]. Figure 26 shows the feedback chain for the 4-bit LFSR and the sequence of values generated by the random function.

```
*************************************************************************
          Genesil Version v7.1 -- Mon Feb 27 21:03:26 1989
Parallel_Datapath: gendavid/davidson/lfsr16          DATAPATH Block Editor
*************************************************************************
                    TST_LAT Functional Specification
                              Block #1

Name:              >tstlatch
Phase X:           PHASE_A      PHASE_B
Shift register:    BASIC        GENERATOR     SIGNATURE
Control signals:   UNCLOCKED    LOCAL         GLOBAL
Polynomial:        >      0x13
Mode:              TRANSPARENT  GATED
Sample:            STDIN_1      STDIN_2
-------------------------------------------------------------------

                                 Connectors:
Load:              >load_____    (LD)
Tin:               >tin______    (TIN)
Tout:              >tout_____    (TOUT)
M1:                >m1_______    (M1)
M2:                >m2_______    (M2)
M3:                >m3_______    (M3)
```

Figure 25. LFSR Genesil Form



TIMEPOINT	LFSR OUTPUT	TIMEPOINT	LFSR OUTPUT
0	iiii	10	0110
1	0001	11	1100
2	0011	12	1001
3	0111	13	0010
4	1111	14	0100
5	1110	15	1000
6	1101	16	0001
7	1010	17	0011
8	0101	18	0111

Figure 26. Feedback Chain for 4-Bit LFSR and Random Vectors

C. SIGNATURE ANALYZERS

The signature analyzer uses the LFSR principles in its operation. In the Genesil implementation, the signature analyzer is essentially an LFSR with its input equal to the output of the circuit or subcircuit to be tested. The particular technique employed is that of a parallel signature

analyzer. In this technique, the outputs of the circuit being tested are connected to the LFSR via XOR gates added between stages in the test register as well as connecting the circuit output to the first LFSR stage [Ref. 2:p. 145]. Figure 27 shows the STL signature configuration for an arbitrary bit position. Note the input mux with additional control inputs of **M4** and **M5**. In the globally and locally clocked options the **M4** and **M5** inputs are generated internally and the device has the same number of external connections as

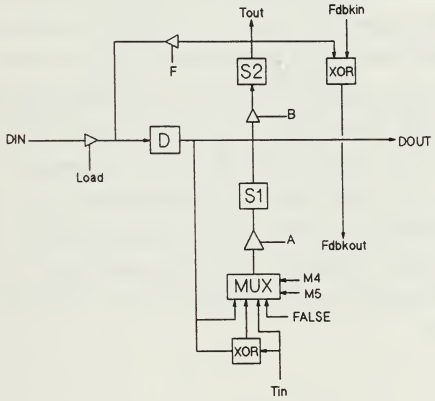


Figure 27. STL Signature Configuration

does the generator function. The XOR gate at the input mux combines the present data value with the value of the preceding shift register stage thus producing a bit-wise checksum value [Ref. 14:p. 24.21].

After a certain number of clock periods, the value in the data latch is a unique value created by the combination of output responses and the XOR feedback chain configuration. If that value differs from the correct value obtained during simulation tests, a fault has been detected. It is obviously important that the LFSR generating the test vectors be intialized to the same starting value for each test and each test run for the same number of clock cycles so that the tests can be repeated. Table 11 shows the inputs and outputs of a simple 8-bit signature analyzer. Note that the designer could design a simple comparator or memory to examine any number of the outputs for a correct response given a certain number of block cycles, thereby moving the analysis function completely onto the chip. The input vectors for this test were produced by an 8-bit LFSR using the default value for the polynomial constant. Each test was run for 30 clock cycles. A maximal length sequence for the LFSR will produce 255 vectors.

TABLE 11
SIGNATURE SIMULATION RESULTS

TIMEPOINT	LFSR OUTPUT	SIGNATURE ANALYZER OUTPUT
31	1110010	11100100
32	11001000	11100100
33	10010000	11100100
34	00100001	00101100 - SIGNATURE
35	01000010	01000010
64	00110001	00110001
65	01100010	00110001
66	11000101	00110001
67	10001010	01010011 - SIGNATURE
68	00010101	0010101
97	01101110	01101110
98	11011101	01101110
99	10111011	01101110
100	01110111	10110011 - SIGNATURE
101	11101110	1101110

This technique is also called compact testing because the output response, after passing through the signature analyzer, can be reduced to a small number of bits. One drawback to the signature analysis technique is due to a phenomenon called **aliasing.** It is possible for a fault to go undetected if its output response produces a signature that is identical to that of a fault-free device. This leads to a loss of fault coverage. Research on this phenomenon has not yet led to the discovery of a simple relationship between the fault coverage and the aliasing phenomenon [Ref. 2:p. 144].

D. IMPLEMENTING BUILT-IN TEST DFT INTO THE CORRELATOR CHIP

The BIT strategy, as opposed to the Scanpath strategy, attempts to facilitate testing by placing the test functions on the chip. Initially, designs were developed that substituted a LFSR where a test register was incorporated in the scanpath designs. This is of limited utility because the LFSR does not produce a custom set of vectors. It requires many more vectors to get the same amount of fault coverage that is achieved with the scanpath using a vector set generated by the ATG program. If the LFSR replaces the test register located in the interior of the circuit in Scanpath Designs #1 and #2, it will not provide any greater fault coverage than did the test registers. The advantages offered by the LFSR are somewhat diminished because a vector set must still be generated for the parts of the circuit located in the front of the LFSR. The LFSR can act as a basic test

register and capture the results of test vectors applied at the inputs to the data registers but the substitution of the LFSR does not enhance the testability of the circuit. The conclusion drawn is that the LFSR is better placed at the front of the circuit where it can generate a stream of vectors at system speed that can be used to test the circuit. This is the approach taken in implementing the BIT strategy into the basic correlator chip.

The first BIT design added a 23-bit LFSR after the primary inputs of the basic correlator chip as shown in Figure 28. In this position, the LFSR can generate test vectors that will include the data bits and the various control inputs. The pseudorandom test vectors propagate through the circuit and the output responses appear on the primary outputs. The first design did not use a signature analyzer.

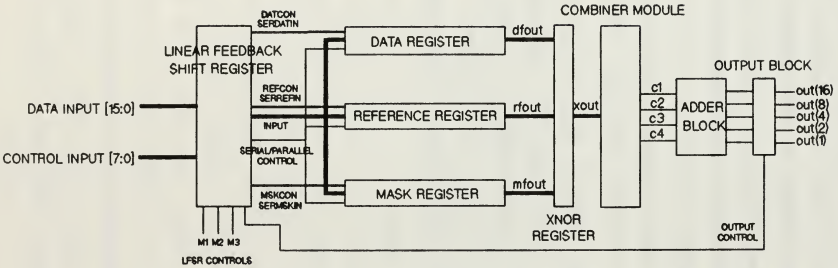


Figure 28. BIT Design #1

When the circuit is to be tested, an initial vector is loaded into the LFSR. This is called the seed vector and is used to initialize the random vector generator. The control inputs are initially set to the sample function discussed in the previous chapter so that the seed vector placed in the data latch is shifted into the serial register. The control inputs are then set to the random function and the LFSR begins to generate vectors. Table 12 shows a sampling of the vectors generated by the LFSR and the values appearing on the output lines.

TABLE 12
SAMPLING OF VECTORS GENERATED BY BIT LFSR

TIMEPNT	testin	testout	cout
1430	LLLLLLLLLLLLLLLLLLLLLLH	010010111101010010001110	00000
1440	LLLLLLLLLLLLLLLLLLLLLLH	100101111010100100011101	00011
1450	LLLLLLLLLLLLLLLLLLLLLLH	001011110101001000111010	00000
1460	LLLLLLLLLLLLLLLLLLLLLLH	010111101010010001110101	00000
1470	LLLLLLLLLLLLLLLLLLLLLLH	101111010100100011101010	00111
1480	LLLLLLLLLLLLLLLLLLLLLLH	011110101001000111010101	00000
1490	LLLLLLLLLLLLLLLLLLLLLLH	111101010010001110101010	00011
1500	LLLLLLLLLLLLLLLLLLLLLLH	111010100100011101010100	00011
1510	LLLLLLLLLLLLLLLLLLLLLLH	110101001000111010101000	00111
1520	LLLLLLLLLLLLLLLLLLLLLLH	101010010001110101010001	00111
1530	LLLLLLLLLLLLLLLLLLLLLLH	010100100011101010100011	00000
1540	LLLLLLLLLLLLLLLLLLLLLLH	101001000111010101000110	00011
1550	LLLLLLLLLLLLLLLLLLLLLLH	010010001110101010001101	00000
1560	LLLLLLLLLLLLLLLLLLLLLLH	100100011101010100011011	01001
1570	LLLLLLLLLLLLLLLLLLLLLLH	001000111010101000110111	00000
1580	LLLLLLLLLLLLLLLLLLLLLLH	010001110101010001101110	00000
1590	LLLLLLLLLLLLLLLLLLLLLLH	100011101010100011011100	00000
1600	LLLLLLLLLLLLLLLLLLLLLLH	000111010101000110111000	00000
1610	*LLLLLLLLLLLLLLLLLLLLLLH*	001110101010001101110000*	00000

The method used to determine the fault coverage with the LFSR configuration involves the use of both the ATG program and the simulation program. The circuit is initialized as discussed above using the simulator and the simulator is then run for a number of cycles. There is no easy way to determine how many vectors need to be generated by the LFSR in order to achieve a certain fault coverage. The output vectors generated as a result of the applied test vectors is captured in a vector file using the Genesil **traceobj** command. This function, when used during simulation, captures the simulation and its results in a data file. Once the simulation is complete, the **untraceobj** command is used and all the vectors created during the simulation are placed in a data file. The newly created data file is then used as an input file for the ATG program. ATG has a function that allows the user to specify an input file and fault grade the vectors in that file.

Table 13 presents the results of a test run using the LFSR. AS seen in the table, the fault coverage does not differ significantly from that of the basic correlator. Also note that the LFSR produced almost four times as many vectors; 2254 vice 605, as did the ATG program when it was used to produce fault coverage for the basic circuit. The designer does not have any control over how many vectors will need to be generated by the LFSR in order to achieve a certain amount of fault coverage. The process is iterative

and only after a number of runs differing lengths, can a judgment be made on how long to make the test sequence. What is significant is that all of the test vectors were produced on the chip and applied to the circuit by the LFSR.

TABLE 13
FAULT COVERAGE COMPARISON

DEVICE	MODULE	VECTORS	TESTS	FAULT COVERAGE
CORLAT_BAS2		605	1215 of 1314	90.47
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		40 of 48	100
LF3CORLAT_BAS2		2254	1213 of 1324	91.61
	ADDER		152 of 170	89.41
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		13 of 15	86.67
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		50 of 58	86.21

The second BIT design developed is shown in Figure 29. The output block has been replaced by a 5-bit signature analyzer. Any number of the 5 primary outputs can be examined to determine the presence of a fault. Table 14 presents a sampling of the vectors produced by the LFSR and the output responses produced by the signature analyzer. The first group of vectors represent the initialization of the circuit. By observing the LFSR output the generation of test vectors can be observed. There is no signature or output at this point because the vectors might not yet generate results. The signature output is obtained from the shift register in the signature analyzer and the output is at the output of the data latches of the analyzer. The second group of vectors show continued generation of test vectors and resultant outputs. The analyzer is in the sample mode of operation so the values in the shift register are the same as those at the output. The third group of vectors represent a signature generation phase. The LFSR has generated over 2000 test vectors at this point and a signature is captured at timepoint 2009. This is indicated by the differing responses on the signature and output lines. At timepoint 2011, the signature is forced onto the output lines. At timepoint 2012, the analyzer is back in sample mode and normal testing resumes. It is clear that nothing can be seen of the functionality of the correlator during the testing operation.

It is always assumed that the chip has been determined to be functionally correct.

Compare these values to the normal outputs shown in Table 13. This approach hardly seems worthwhile when dealing with a small number of output lines in a circuit increase and a significant effort is required to detect a fault. Analysis can be done much more easily and with less cost when the set of outputs to be checked is reduced.

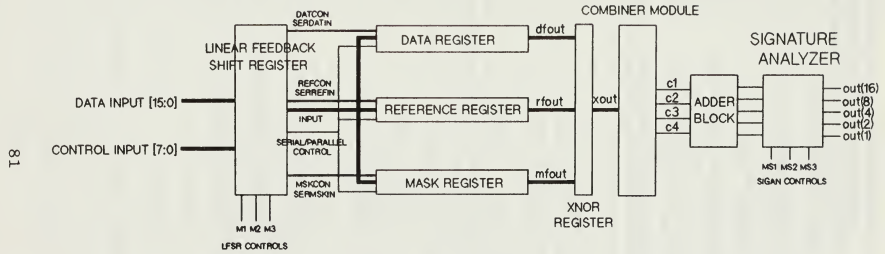


Figure 29. BIT Design #2

TABLE 14
SIMULATION SAMPLING OF BIT DESIGN #2

TIMEPOINT	LFSR OUTPUT	SIGNATURE	OUTPUT
0	iiiiiiiiiiiiiiiiiiiii	iiiii	iiiii
1	00000000000000000000001	iiiii	iiiii
2	00000000000000000000001	iiiii	iiiii
3	00000000000000000000010	iiiii	iiiii
4	00000000000000000000100	iiiii	iiiii
5	00000000000000000001000	iiiii	iiiii
24	10000100001000010000100	00011	00011
25	00001000010000100001001	00011	00011
26	00010000100001000010010	00011	00011
27	00100001000010000100101	00010	00010
2007	11101101110010100011100	01000	01000
2008	11011011100101000111000	00101	00101
2009	10110111001010001110000	01111	00101
2010	01101110010100011100000	01111	00101
2011	11011100101000111000000	01111	01111
2012	10111001010001110000001	00110	00110
2013	01110010100011100000011	00100	00100

V. AUTOMATIC TEST GENERATION

A. THE AUTOMATIC TEST GENERATION PROGRAM

The Automatic Test Generation (ATG) program used to generate test vectors and fault grade BIT designs is an optional tool available with the Genesil Silicon Compiler. It was loaned to the Naval Postgraduate School VLSI Laboratory by Silicon Compiler Systems Corporation to be used in developing and evaluating the DFT strategies described in this project.

ATG was designed to uncover manufacturing defects in a completed chip. The faults are modeled as "stuck-at" faults as described in the first chapter. ATG examines the designed circuit before manufacturing and produces vectors capable of detecting as many faults as possible within the constraints of the algorithms used in this program. It is important to stress that ATG provides the detection not the location of the fault.

ATG is especially valuable because it provides the designer with quick feedback on the testability of a circuit. It is most effectively used in the early stages of the design. By running ATG on each section as it is produced, areas that reveal themselves as untestable can be redesigned early in the design process or test registers can be included to enhance the observability and controllability of the internal nodes. The sets of test vectors can be saved and

used to check the chip after it returns from the manufacturer.

The algorithm used in the ATG program is a version of a classical algorithm called the D-algorithm. Developed in 1966 by J.P. Roth [Ref. 20], the D-algorithm provides a calculus to compute tests for failures. The algorithm defines a failure as a transformation of hardware that changes the logical functioning of the circuit. It defines a primary output as a line that is not fed by any other lines in the circuit and a primary output as a line whose signal is accessible to the outside of the circuit. Finally, it defines a test as a pattern of signals on the primary inputs that produces a response on the primary outputs whose value differs in the presence of a failure. [Ref. 20:p. 278]

Roth developed a five-valued calculus that carries out line **sensitization** and **justification.** Sensitization is closely related to observability. Sensitization is the process by which the algorithm propagates a value to the primary outputs. If the value can be propagated without any conflicts, then the path is said to be sensitized and the node tested is observable. Justification is closely related to controllability. If the values required to test a gate can be backed through the circuit to the primary inputs, then the test is said to be justified and the nodes at the input of the gate to be tested are said to be controllable.

ATG uses justification to generate the test inputs and sensitization to check the outputs [Ref. 15:p. 1.3]. Genesil has forty primitive elements that make up the various blocks and modules that are available to the designer. They vary from a simple AND gate to a complicated tristate net. Each primitive element has its own justification and sensitization models. In applying the D-algorithm, values are placed on the inputs of the gate being tested and then backed through the circuit to the primary inputs. If the values can be placed on the primary inputs without any conflicts, all is well. If a conflict develops and the values cannot be produced at the primary inputs, the process is backtraced to another spot and ATG takes a different path to the primary inputs and the process is repeated. The same is true of the sensitization process. ATG tries to find a path to propagate the M or W value to the primary output. The M or W values refer to the value on the output of the gate being tested. Again, if a certain path produces a contradiction, ATG tries a different path. If either process fails to produce a satisfactory solution, the fault is determined to be untestable.

One of the areas that is very difficult to handle when developing a test strategy is sequential logic. ATG attempts to facilitate sequential testing by translating the sequential circuits into combinational logic circuits that exist over a limited time range. The technique is called

time **unrolling**. Simply put, ATG treats each sequential element as a multiplexer. When the clock is HIGH, the element selects its data input; when the clock is LOW, the element selects the output at the previous timepoint. Sequential elements are thus reduced to a "stack" of elements. Each element is a copy of the original element and its position in the stack is determined by its timepoint. Figure 30 shows how ATG would treat a simple latch element [Ref. 15:p. 1.6].

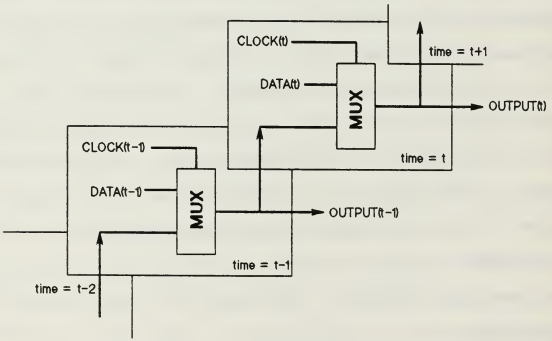


Figure 30. ATG Time Unrolling for a Simple Latch

In conducting a test ATG first determines which faults are obviously untestable. Examples of untestable faults are gates with an input tied to a constant value or gates whose outputs are ignored. It also determines which nodes are most easily controllable and observable. These determinations will help ATG make choices when it reaches a decision point in finding paths of justification or sensitization. Rather than working on only one path to the primary inputs and outputs at a time, ATG uses what is called a modified breadth-first search. This search proceeds gradually to the periphery of the circuit, working on all signals and paths. The program takes one step in processing a particular path, puts the resulting justification or sensitization onto a list of other pending processes, and then goes onto the next process. In this manner, the path from the gate being tested expands to the primary inputs and outputs.

The breadth-first approach allows conflicts between different paths of assertion to be recognized and corrected early, speeding up the test generation. The disadvantage, however, is that when a conflict is found, ATG backs up until it finds the assertion that created the conflict and repeats all justification and sensitization processes between the point where the conflict was discovered and its cause. This often requires that paths that are not related to the conflicts are also redone [Ref. 15:p. 1.7].

B. THE ATG FORM

Figure 31 shows the ATG form that the designer works with when testing is being done. It consists of two parts, a control part and a status part. The control section allows the designer to set certain parameters and include additional vector files into the testing process.

- The **Output File** specifies where the generated test vectors are to be written.
- The **Sequential Depth** refers to the number of phases that are required to pass a signal from primary input to output. The default value is set to the maximum number of latches found in the paths between input and output.
- The user can specify how many **Random Input Vectors** are to be used as seed vectors. ATG generates the random vectors in an attempt to speed up the testing process.
- The **Simulation Vectors** field refers to the number of vectors that will be used to initialize a circuit. ATG will fault grade the remaining test vectors.
- The **Default Toggles** field refers to the clocks to be used.
- The user can elect to limit the amount of CPU time to be used to generate a test vector file. The default value is 1800 seconds. The designer can select NO and ATG might run for hours. It is most useful to start with a small time and work upwards.
- Input files can be used and fault graded by selecting YES in the **Fault Grade Only** field. If the default is selected (NO), then ATG will continue to generate new vectors and determine fault coverage even after an input file's entries have been exhausted.
- The **Enable Input File** field is the location to enter the name of a file that the user wishes to provide test vectors. This was the approach used in determining the fault coverage for the BIT design discussed earlier.
- The **Enable Init File** provides a method for the designer to modify the clocks to be used in simulation.

- The **Enable Coverage In** and **Default Coverage Out** refers to the method that the ATG program uses to keep track of which faults have been tested. These options allow the designer to use coverage maps produced during earlier runs to reduce the amount of retesting that takes place during additional test runs.

```
*******************************************************************************
            Genesil Version v7.1 -- Thu Mar  2 20:55:08 1989
Module: ~gendavid/davidson/corlat_bas2                    ATG Control Program
*******************************************************************************
                                ATG Control

    Output File:              >vecs

    Sequential Depth:         >-1
    Random Input Vectors:     >0
    Initialization Vectors:   >0
    Default Toggles:           NO YES
    Limit Time:                NO YES
    Time Limit:               >1800
    Limit Coverage:            NO YES
    Fault Grade Only:          NO YES
    Enable Input File:         NO YES
    Enable Startup File:       NO YES
    Enable DFT File:           NO YES
    Enable Coverage In:        NO YES
    Default Coverage Out:      NO YES

                                ATG Status

    Vector          Tests                        CPU Time (h:m:s)
    ------  ------------------------------  ------------------
            Change      Tested      Percent     Change      Total

    336     0           1199        92.30       0           30:07
    337     0           1199        92.30       1           30:08

                              Command Status

    ATG done
```

Figure 31. Automatic Test Generation Form

The second part of the ATG form is the status section. The status section provides information about the current test. It displays the number of the last two vectors generated under the vector heading. Under the tests heading, information is presented telling how many tests were instantiated by each vector, the total number of tests instantiated so far and the fault coverage percentage so far. Finally, under the CPU heading, ATG displays how much CPU time is used for each vector and how much total CPU time has been used for the test so far.

There are two ways to update the status section once a test is running, the **UPDATE_SCREEN** command from the ATG menu commands can be used. This command will update the status section to provide the most current information. It does not provide a constant updating function. To continually update the screen, the command **update_loop** can be entered on the prompt line. This command will provide constant updating of the status section until the test run is complete or until the user aborts the command.

C. ANALYZING ATG RESULTS

Besides providing feedback via the status section of the ATG form, ATG supplies specific feedback concerning areas that were or were not covered during the test run. The **ANALYZE** command of the ATG menu allows the designer to hierarchically examine the circuit to find specifically what areas had good and poor fault coverage. Figure 32 contains a

portion of a Genesil session log that shows an example of the hierarchical breakdown of the circuit used in the combiner block. As discussed in Chapter III, certain gates in the combiner were not able to be observed due to conflicts at the primary outputs. Figure 33 shows, at the gate level, the specific tests that were not instantiated. From this information, the block was redesigned as described in Chapter III.

```
atg
run_atg
)  Checking file currency . . .
) Internal Object Hierarchy Initialized. . . .
) Completing Data Gathering Phase . . . .
) All files are up to date.
)  Running ATG in background
update_loop
)  ATG begun
)  ATG is DONE
ACCEPT_FORM
)  Form is valid
)  Text is written
atg
ANALYZE
) ap_prune_default
FAULTS
) / (module):  347 tests out of 394 (88.0711%)
)    combiner (module):  160 tests out of 192 (83.3333%)
)       combin3 (module):  40 tests out of 48 (83.3333%)
)          XOR4 (module):  3 tests out of 4 (75%)
)             xor (XOR):  3 tests out of 4 (75%)
)          XOR5 (module):  1 tests out of 4 (25%)
)             xor (XOR):  1 tests out of 4 (25%)
)          OR0 (module):  2 tests out of 3 (66.6667%)
)             or (OR):  2 tests out of 3 (66.6667%)
)          OR1 (module):  2 tests out of 3 (66.6667%)
)             or (OR):  2 tests out of 3 (66.6667%)
)          AND4 (module):  2 tests out of 3 (66.6667%)
)             and (AND):  2 tests out of 3 (66.6667%)
)          AND5 (module):  2 tests out of 3 (66.6667%)
)             and (AND):  2 tests out of 3 (66.6667%)
)       combin2 (module):  40 tests out of 48 (83.3333%)
)          XOR4 (module):  3 tests out of 4 (75%)
)             xor (XOR):  3 tests out of 4 (75%)
)          XOR5 (module):  1 tests out of 4 (25%)
)             xor (XOR):  1 tests out of 4 (25%)
)          OR0 (module):  2 tests out of 3 (66.6667%)
```

Figure 32. ATG Analysis Showing Heirarchical Breakdown

```
sel
AND4
sel abd
) No such path abd
sel and
faults
) and (AND):  2 tests out of 3 (66.6667%)
)   Pin 0 /combiner/combin0/XOR1/xor (XOR) (gate 304)
)   Pin 1 /$dummy/$GA24 (JU1_A1) (gate 24)
) Pattern  result
)    11  UNTESTED
)    01  TESTED
)    10  TESTED
sel ..
sel ..
sel
AND5
sel and
faults
) and (AND):  2 tests out of 3 (66.6667%)
)   Pin 0 /$dummy/$GA21 (JU1_A1) (gate 21)
)   Pin 1 /combiner/combin0/XOR0/xor (XOR) (gate 305)
) Pattern  result
)    11  UNTESTED
)    01  TESTED
)    10  TESTED
sel ..
sel ..
sel
OR0
sel or
faults
) or (OR):  2 tests out of 3 (66.6667%)
)   Pin 0 /combiner/combin0/XOR5/xor (XOR) (gate 300)
)   Pin 1 /combiner/combin0/XOR2/xor (XOR) (gate 303)
) Pattern  result
)    00  TESTED
)    10  UNTESTED
)    01  TESTED
sel ..
```

Figure 33. ATG Analysis Showing Gate Level Breakdown

VI. CONCLUSIONS

A. SUMMARY

This thesis has examined two of the main Design for Testability techniques being practiced today as they are implemented by the Genesil silicon compiler, the Scanpath Design techniques and the Built-in Test technique. The use of the Shiftable Test Latch in a variety of configurations provides the designer with great flexibility in implementing an efficient testing scheme and does not appreciably increase the number of pins required. The addition of the ATG program provided a tool by which fault coverage for the different implementations could be evaluated.

The Scan Path technique had a number of advantages which led to an optimum amount of fault coverage. Among these advantages are:

- Scanpaths enhance the observability and controllability of difficult-to-reach internal nodes. By increasing the accessibility of these nodes, fault coverage can be increased. This was demonstrated in Chapter III by the redesign of the combiner module and the inclusion of a test register.

- The set of test vectors generated by the scanpath is customized to the circuit under test, consequently it is smaller than the set of test vectors required by other DFT techniques such as Built-in Test. This is demonstrated by the results of Table 15. The table compares the fault coverage given by the correlator chip without any DFT measures, with a scanpath, and using an LFSR to generate test vectors. The scanpath design achieved the highest fault coverage and used the smallest number of test vectors.

- Scanpaths allow the designer to have absolute control over the vectors to be used in testing. It also provides a method to observe the state of the internal nodes during normal operation. This has the added advantage of changing sequential sections of the design into combinational logic by the addition of an STL in the feedback loop.

There are also disadvantages to the Scan Path technique:

- An outside tester is required to apply the test vectors and analyze the responses. As circuits become more complex, testing becomes more difficult and testers more expensive.

- The fact that test vectors must be loaded serially into the shift registers increases testing time. Genesil has attempted to help alleviate this problem by providing a function in the STL that allows new values to be loaded into the shift register as test vectors are loaded in the data latches. The inclusion of a local test clock allows values to be shifted into the test register without interfering with the normal operation of the system or being delayed by the system clock.

TABLE 15
FAULT COVERAGE COMPARISONS

DEVICE	MODULE	VECTORS	TESTS	FAULT COVERAGE
CORLAT_BAS2		605	1215 of 1314	90.47
	ADDER		154 of 170	90.59
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		40 of 48	100
TGCORLAT_BAS3		133	1310 of 1378	95.07
	ADDER		164 of 170	96.47
	COMBINER1		112 of 112	100
	COMBINER2		80 of 80	100
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		15 of 15	100
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		101 of 112	90.18

TABLE 15 (cont)

LF3CORLAT_BAS2		2254	1213 of 1324	91.61
	ADDER		152 of 170	89.41
	COMBINER		160 of 192	83.33
	DATA_IN		242 of 259	93.44
	MASK_IN		242 of 259	93.44
	OUTPUT		13 of 15	86.67
	REF_IN		242 of 259	93.44
	XNORREG		112 of 112	100
	$DUMMY		50 of 58	86.21

The second method to be evaluated was the Built-in Test technique. The advantages of the BIT strategy include:

- Some or all of the test functions can be moved onto the chip. The first BIT design examined the circuit with only a linear feedback shift register added for test generation. The second design included both an LFSR and a signature analyzer for generation and analysis. The inclusion of the test functions within the chip help to decrease the cost of testing and allow tests to be run with a minimum of outside involvement.

- Tests can be generated and run at system speed. This allows potentially millions of tests to be performed each second. The time of testing can be significantly reduced from that of the scanpath designs.

- The use of the signature analyzer greatly eases the burden of outside analysis of testing. While it does not remove entirely the need to examine the responses it reduces the number of output responses that need to be examined.

- Genesil gives the designer ready-made generators and signature analyzers with defaults optimized to the testing problem. However, the designer can reconfigure

the feedback chains used in these devices if a specific application is desired.

There are disadvantages to the BIT techniques:

- The user has no control over the sequence of test vectors to be applied to the circuit under test other than selecting the polynomial constants and starting values.

Each different DFT strategy has its pros and cons. The most suitable implementation of DFT might include both types of DFT strategies within a single chip. An LFSR is used to generate strings of vectors to apply to test the nodes that are easily observable and controllable. Scanpaths are used to provide a method to enhance the observability and controllability of nodes that might otherwise be untestable. The signature analyzer can be included at the output of the circuit to compact the output response of the circuit during testing.

The methodology that should be employed includes the testing and accurate simulation of every module as it is designed. In this manner, sections which exhibit poor testability can be redesigned before actual production. As the circuit is being built up and different modules and blocks are being combined, they also should be tested and simulated. Finally, the entire chip is assembled and final testing is completed. The testing done at this level should be retained and provide after-manufacture testing.

B. RECOMMENDATIONS

The following are recommendations for further study:

1. The continued development of the use of the silicon compiler as a tool for system designers without in-depth knowledge of VLSI design techniques. The automatic capabilities of the Genesil silicon compiler are particularly attractive to a designer attempting the rapid development and testing of a custom system.

2. Build a larger chip than the correlator chip developed in this research, implement the Design for Testability strategies described in this research, and continue the project through manufacture and testing.

3. Develop a set of translation programs to allow vector files generated on the silicon compiler to be used by the TEKTRONIX tester available in the VLSI laboratory. This will allow the testing of Genesil-designed chips with existing hardware.

4. Research the use of linear feedback shift registers in testing and methods to optimize the polynomials chosen to implement the XOR feedback paths.

APPENDIX

DESIGN FOR TESTABILITY TUTORIAL

A. INTRODUCTION

The purpose of this tutorial is to introduce the designer to the shiftable test latch and the different configurations that are included in the Genesil silicon compiler. It will demonstrate to the user the various forms that are required for the successful creation of a test register as a random logic block or a parallel datapath. Prior to beginning any work on Genesil, a user is encouraged to read and become familiar with the tutorial included in Robert Settle's thesis, "Design Methodology Using the Genesil Silicon Compiler" [Ref. 5]. It provides a good guide to the mechanics of the Genesil hierarchical approach to chip design. This tutorial is written with the assumption that the user has already read Reference 5 and is familiar with the basic manipulation of Genesil.

It is important to stress that all designs must be carefully replanned; a sketch is very useful. Because Genesil relies heavily on a netlist description of the circuit, the designer must ensure that all net names are unambiguous and complete.

B. RANDOM LOGIC BLOCK

The first implementation to be examined is that of the test register comprised of a cascaded number of stages of the shiftable test latch (STL). The initial design will be an 8-bit test register using the globally clocked STL. Figure A.1 shows the random logic block specification form for the STL before any of the fields have been completed. Note that in the global implementation all of the input (control or data) are controlled by the two clock phases PHX and PHY. Figure A.2 shows the form after it has been completed. Notice that the width field is now set to 8 and the DIN and DOUT connectors have been given signal names in bus notation representing 8 bit positions. The inputs **m1** and **m2** are signals that are decoded in order to provide the correct sequence of control inputs as shown in Table A.1 [Ref. 13:p. 15.14]. Figure A.3 shows the Genesil icon for the 8-bit test register. These forms are relatively easy to complete, but the designer must ensure that the right signal names are given to the input and output lines in order to make proper connections to the rest of the circuit.

Figure A.4 shows the implementation of the STL with a local clock scheme, again note the clocked inputs and outputs and the additional test clock (**phase_ta** and **phase_tb**). The local clock scheme requires clock inputs independent of the system clock. This version of the STL allows serial data to the shifted in or out independently of normal operation. Its

table of operation is the same as that used for the globally clocked STL.

Figure A.5 shows the form used for the unclocked version of the STL. Note here that the **a**, **b**, **f**, and **s** inputs are all strobed inputs. The unclocked STL's controls operate independently of any system clock. These strobed signals typically are generated off the chip and it is up to the user to ensure that they are properly timed to prevent any contention with each other and normal operation. The signals to be used and their proper sequence are shown in Table A.2 [Ref. 13:p. 15.6].

C. PARALLEL DATAPATH

The second implementation of the STL that Genesil offers is in a parallel datapath. The basic STL can be constructed as well as the vector generator mode and the signature analyzer mode. Figure A.6 shows the Genesil icon used to represent the STL. The parallel datapath configuration is made of three components, an interface on the left or input side, the test latch in the middle, and a general purpose port on the right or output side. Figure A.7 shows the specification form creating the datapath. Note that it is 8 bits wide and uses a direct driver. It is helpful to note that if the bus name is left in its default name (BUS_B in this case), the bus will act as a feedthrough.

Once the basic datapath has been established, blocks must be added to it. The first block added is the interface

block. The specification form for the interface block, before any definition has occurred, is shown in Figure A.8. Figure A.9 shows the form after is has been completed. The interface shown here gets its input from Bus A (input steering) and Bus B is not connected. Note also that for its output steering, it drive **Stdout** which will continue the parallel datapath to the next component, the test latch.

The test latch specification form is shown in Figure A.10. From this form the test latch can be configured in 9 different ways. This particular form is set up in the basic unclocked mode. Other than the extra work required to complete the other blocks of the parallel datapath, this implementation is essentially the same as that used in the random-logic basic testlatch.

Figure A.11 shows the specification form used for the general purpose output port. The important things to notice with this form are the variety of speed/power combinations the designer can choose and the mask that specifies which connectors are to be used. This is a hexadecimal number and a 1 in any bit position signifies that there is to be a connector there. Figure A.12 shows the completed Datapath Functional Specification Form.

Figure A.13 shows the specification form for the STL using the vector generate mode of operation. The only added field is that representing the polynomial constant discussed in Chapter IV. When the form is first entered, the

polynomial value is set automatically based on the width of the datapath. However, if the width is changed, the polynomial is not also changed. It is important to check the system manual for an optimized polynomial and input it as desired. The signature analyzer form is very similar to the vector generator.

This overview should allow the designer to quickly implement his own DFT strategies. Detailed information on each of the types of STL configurations is included in the Genesil System manual [Ref. 14 and 15] and a good overview of the capabilities of Genesil and DFT are included in the Johannsen article, "Genesil Silicon Compilation and Design for Testability" [Ref. 12].

```
*******************************************************************************
                Genesil Version v7.1 -- Thu Mar  2 21:23:06 1989
Random_Logic: ~gendavid/davidson/dft_tst                Random Logic Block Editor
*******************************************************************************
                          Random Logio Block Specification

Block type:  TLATCHG
Block index: 0
Name:        >TLATCHG0
Width:       >1

                Regime
Connector  Width  | Timing
PHX          1      1 Phase X     >FALSE
PHY          1      1 Phase Y     >FALSE
TIN          1      1 Vx(t)       >FALSE
TOUT         1      1 Sx(t+1)     >NC
M1           1      1 Vy(t-1)     >FALSE
M2           1      1 Vy(t-1)     >FALSE
LOAD         1      1 Vy(t-1)     >FALSE
DIN          1      1 Vx(t)       >FALSE
DOUT         1      1 Sy(t)       >NC
```

Figure A.1. STL Random Logic Block Specification Form

```
*******************************************************************************
                Genesil Version v7.1 -- Thu Mar  2 21:26:03 1989
Random_Logic: ~gendavid/davidson/dft_tst                Random Logic Block Editor
*******************************************************************************
                          Random Logic Block Specification

Block type:  TLATCHG
Block index: 0
Name:        >TLATCHG0
Width:       >8

                Regime
Connector  Width  | Timing
PHX          1      1 Phase X     >phase_a
PHY          1      1 Phase Y     >phase_b
TIN          1      1 Vx(t)       >tin
TOUT         1      1 Sx(t+1)     >tout
M1           1      1 Vy(t-1)     >m1
M2           1      1 Vy(t-1)     >m2
LOAD         1      1 Vy(t-1)     >load
DIN          8      1 Vx(t)       >din[7:0]
DOUT         8      1 Sy(t)       >dout[7:0]
```

Figure A.2. Completed STL Specification Form

TABLE A.1
TRUTH TABLE OPTIONS FOR LOCAL/GLOBAL CONFIGURATION

Encoded Inputs M1 M2	Operation	Decoded Outputs PHASE X	PHASE Y
0 0	SHIFT	A	B
1 0	FORCE (1)	F	B
0 1	SAMPLE	S	B
1 1	SWAP (2)	S	-

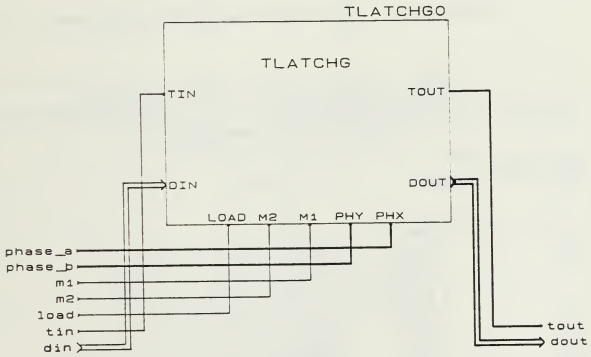


Figure A.3. Genesil Icon for 8-bit
Random Logic Test Register

```
*************************************************************************
          Genesil Version v7.1 -- Thu Mar  2 21:41:37 1989
Random_Logic: ~gendavid/davidson/dft_tst          Random Logic Block Editor
*************************************************************************
                    Random Logic Block Specification

Block type:  TLATCHL
Block index: 1
Name:        >TLATCHL1
Width:       > 8

                 Regime
Connector  Width | Timing
PHX          1     1 Phase X    >phase a
PHY          1     1 Phase Y    >phase b
PHTA         1     2 Phase X    >phase ta
PHTB         1     2 Phase Y    >phase tb
TIN          1     2 Vx(t)      >tin
TOUT         1     2 Sx(t+1)    >tout
M1           1     2 Vy(t-1)    >m1
M2           1     2 Vy(t-1)    >m2
LOAD         1     1 Vy(t-1)    >load
DIN          8     1 Vx(t)      >din[7:0]
DOUT         8     1 Sy(t)      >dout[7:0]
```

Figure A.4. STL with Local Clocking Scheme

```
*************************************************************************
          Genesil Version v7.1 -- Thu Mar  2 21:45:23 1989
Random_Logic: ~gendavid/davidson/dft_tst          Random Logic Block Editor
*************************************************************************
                    Random Logic Block Specification

Block type:  TLATCHU
Block index: 0
Name:        >TLATCHU0
Width:       > 8

                 Regime
Connector  Width | Timing
PHX          1     1 Phase X    >phase a
PHY          1     1 Phase Y    >phase b
A            1     2 Strobe Vx  >a
B            1     3 Strobe Vx  >b
F            1     4 Strobe Vx  >f
S            1     5 Strobe Vx  >s
TIN          1     2 Vx(t)      >tin
TOUT         1     3 Sy(t)      >tout
LOAD         1     1 Vy(t-1)    >load
DIN          8     1 Vx(t)      >din[7:0]
DOUT         8     1 Sy(t)      >dout[7:0]
```

Figure A.5. Unclocked Version of STL

TABLE A.2
TRUTH TABLE OPTIONS FOR UNCLOCKED CONFIGURATION

Inputs A B F S	Operation
0 0 0 0	HOLD
1 0 0 0 0 1 0 0	SHIFT *
0 0 1 0	FORCE
0 0 0 1 0 1 0 0	SAMPLE *
0 0 0 1 0 0 1 0 0 1 0 0	SWAP *

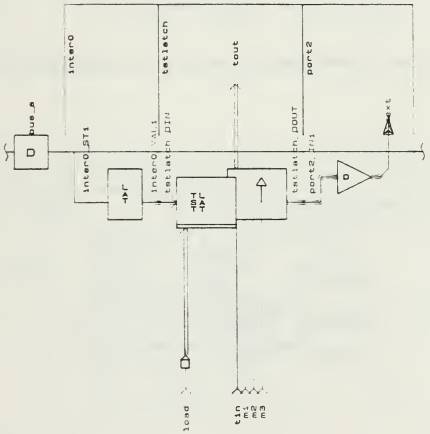


Figure A.6. Genesil Icon for 8-bit Parallel Datapath Configuration

```
**************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:49:50 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2            DATAPATH Block Editor
**************************************************************************
                      DATAPATH Functional Specification:
Width:               > 8
Phase A:             >phase a_____.   (PHASE_A)
Phase B:             >phase b_____    (PHASE_B)
--------------------------------------------------------------------------
Bus A Name:>bus a
Driver:              DIRECT      PRECHARGE    TRISTATE      NONE
Out Left: YES  NO              Out Right:YES  NO
--------------------------------------------------------------------------
Bus B Name:>BUS B              Transfer:  PHASE A  PHASE_B
Driver:              DIRECT      PRECHARGE    TRISTATE      NONE
Out Left: YES  NO              Out Right:YES  NO
--------------------------------------------------------------------------
Number of slices:    > 0
--------------------------------------------------------------------------
                                    Blocks:
```

Figure A.7. Specification Form for Parallel Datapath

```
**************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:50:24 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2            DATAPATH Block Editor
**************************************************************************
                      INTERFACE Functional Specification
                                Block #0

Name:                >INTER0
--------------------------------------------------------------------------
                            Standard Input/Output 1:
Top View:            NO CONNECT   THROUGH      LOGIC_FUNCTIONS
--------------------------------------------------------------------------
                            Standard Input/Output 2:
Top View:            NO CONNECT   THROUGH      LOGIC_FUNCTIONS
```

Figure A.8. Specification form for Interface Block

```
**************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:51:14 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2          DATAPATH Block Editor
**************************************************************************
                    INTERFACE Functional Specification
                                Block #0

Name:                >inter0

-------------------------------------------------------------
                          Standard Input/Output 1:
Top View:            NO_CONNECT    THROUGH       LOGIC_FUNCTIONS
Phase X:             PHASE_A       PHASE_B

Input Steering:      STDIN_1       STDIN_2       BUS_A         BUS_B
                     CONSTANT      EXTERNAL
                     STD_MUX       BUS_MUX       2_WAY_MUX     4_WAY_MUX

Input Inversion:     NONE          CONSTANT      SELECTABLE

Shift:               NO        LEFT       RIGHT      BOTH

Latch:               NONE          TRANSPARENT   GATED         DFF
                     LEFT          RIGHT         BOTH

Output Steering:
Drive Stdout:        YES           NO
Drive Bus A:         YES           NO
Drive Bus B:         YES           NO
-------------------------------------------------------------
                          Standard Input/Output 2:
Top View:            NO_CONNECT    THROUGH       LOGIC_FUNCTIONS
```

Figure A.9. Completed Interface Block Form

```
**************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:53:14 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2          DATAPATH Block Editor
**************************************************************************
                    TST_LAT Functional Specification
                                Block #1

Name:                >tstlatch
Phase X:             PHASE_A       PHASE_B
Shift register:      BASIC         GENERATOR     SIGNATURE
Control signals:     UNCLOCKED     LOCAL         GLOBAL
Mode:                TRANSPARENT   GATED
Sample:              STDIN_1       STDIN_2
-------------------------------------------------------------
                                   Connectors:
Tin:                 >tin          (TIN)
Tout:                >tout         (TOUT)
A:                   >a            (A)
B:                   >b            (B)
F:                   >f            (F)
S:                   >s            (S)
```

Figure A.10. Datapath Basic STL Specification Form

```
********************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:53:58 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2          DATAPATH Block Editor
********************************************************************************
                      GENERAL PORT Functional Specification
                                  Block #2

Name:                 >port2___
--------------------------------------------------------------
                                Standard Input/Output 1:
Top View:             NONE      PASS      IN_PORT   OUT_PORT
Connector Name:       >ext_____________   (EXT1)
Fan:                  TOP          BOTTOM       SPLIT
Mask:                 >0xffffffff

Speed/Power:          HIGH_SPEED   AVERAGE      LOW_POWER    NO_BUFFER
Driver:               DIRECT    TRISTATE  PRECHARGED
--------------------------------------------------------------
                                Standard Input/Output 2:
Top View:             NONE      PASS      IN_PORT   OUT_PORT
```

Figure A.11. Specification Form for General Purpose Port

```
********************************************************************************
              Genesil Version v7.1 -- Thu Mar  2 21:54:15 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2          DATAPATH Block Editor
********************************************************************************
                        DATAPATH Functional Specification:
Width:                >_8
Phase A:              >phase a_________   (PHASE_A)
Phase B:              >phase b_________   (PHASE_B)
--------------------------------------------------------------
Bus A Name:>bus a_____________
Driver:               DIRECT       PRECHARGE     TRISTATE      NONE
Out Left: YES  NO                Out Right:YES  NO
--------------------------------------------------------------
Bus B Name:>BUS B_____________    Transfer:  PHASE A  PHASE B
Driver:          DIRECT           PRECHARGE     TRISTATE      NONE
Out Left: YES  NO                Out Right:YES  NO
--------------------------------------------------------------
Number of slices:   >_0
--------------------------------------------------------------
                                    Blocks:
COPY DEL EDIT FLIP MOVE     0: >inter0___  (interface)
COPY DEL EDIT FLIP MOVE     1: >tstlatch   (testability latch)
COPY DEL EDIT FLIP MOVE     2: >port2____  (general port)
```

Figure A.12. Completed Specification Form for Parallel Datapath

```
*******************************************************************************
                Genesil Version v7.1 -- Thu Mar  2 21:56:53 1989
Parallel_Datapath:  gendavid/davidson/dft_tst2              DATAPATH Block Editor
*******************************************************************************
                          TST_LAT Functional Specification
                                    Block #1

Name:                 >tstlatch
Phase X:              PHASE_A       PHASE_B
Shift register:       BASIC         GENERATOR     SIGNATURE
Control signals:      UNCLOCKED     LOCAL         GLOBAL
Polynomial:           >     0x11d
Mode:                 TRANSPARENT   GATED
Sample:               STDIN_1       STDIN_2
-------------------------------------------------------------------------------
                                    Connectors:
Tin:                  >tin                  (TIN)
Tout:                 >tout                 (TOUT)
PHASE_TA:             >phase_ta             (PHASE_TA)
PHASE_TB:             >phase_tb             (PHASE_TB)
M1:                   >m1                   (M1)
M2:                   >m2                   (M2)
M3:                   >m3                   (M3)
```

Figure A.13. Specification Form for Datapath STL

LIST OF REFERENCES

1. Tsui, F.F., LSI/VLSI Testability Design, McGraw-Hill Book Company, 1987, pp. 1-98.

2. Pradhan, D.K., ed. Fault-Tolerant Computing, Theory and Techniques, 2 vols. New Jersey:Prentice-Hall, 1986, pp. 1-153.

3. Wadsak, P.L., "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", Bell System Technical Journal, vol. 57, pp. 1449-1474, May-June 1978.

4. Payne, D., "Silicon Compilation in ASIC Design", Defense Computing, vol. 1, no. 6, pp. 38-40, November-December 1988.

5. Settle, R.H., Design Methodology Using the Genesil Silicon Compiler, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1988.

6. Rockey, R.R., Silicon Compiler Implementation of a Kalman Filter Algorithm as an ASIC, Master's Thesis, Naval Postgraduate School, Monterey, December 1988.

7. Beck, T. and Galinis, W., "16-Bit VISI Correlator Chip Design," Unpublished Report, March 1988.

8. Jones, N.B., Digital Signal Processing, Peter Peregrinus Ltd., United Kingdom, 1982.

9. Genesil System, Timing Analysis User's Guide, Silicon Compiler Systems Corp., San Jose, California, July 1987.

10. Genesil System, Simulation User's Guide, Silicon Compiler Systems Corp., San Jose, California, September 1987.

11. Genesil System, GENIE User's Guide, Silicon Compiler Systems Corp., San Jose, California, February 1988.

12. Johannsen, D. and Sabo, D., "Genesil Silicon Compilation and Design for Testability", 3rd International IEEE VLSI Multilevel Interconnection Conference, pp. 372-380, 1986.

13. Genesil System, Volume III, Random Logic Module, Silicon Compiler Systems, San Jose, California, September 1987.

14. Genesil System, Volume II, Parallel Data Module, Silicon Compiler Systems, San Jose, California, September 1986.

15. Genesil System, ATG User's Guide, Silicon Compiler Systems, San Jose, California, September 1988.

16. McCluskey, E.J., "Built-In Self-Test Techniques," IEEE Design and Test, April 1985.

17. Williams, T.W. and Parker, K.P., "Design for Testability - A Survey," Tutorial: VLSI Testing & Validation Techniques; reprints of key papers, IEEE Computer Society Press, 1985.

18. Golumb, S.W., Shift Register Sequences, Aegean Park Press, Laguna Hills, California, 1982.

19. Peterson, W.W. and Weldon, E.J., Error-Correcting Codes, The MIT Press, Cambridge, Massachusetts, 1972.

20. Roth, J.P., "Diagnosis of Automata Failures: A Calculus and a Method," IBM Journal of Research and Development, vol. 10, pp. 278-291, July 1966.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	1
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 62 Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curricular Officer, Code 32 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Prof. C. Yang, Code 62Ya Naval Postgraduate School Monterey, California 93943-5000	2
6.	Prof. H. Loomis, Jr., Code 62Lm Naval Postgraduate School Monterey, California 93943-5000	5
7.	Commander, Naval Research Laboratory ATTN: Mr. Andy Fox, Code 8120 4555 Overlook Ave., S.W. Washington, DC 20375	1
8.	Commander, Naval Research Laboratory ATTN: LT Brian Kosinski, Code 9110 4555 Overlook Ave., S.W. Washington, DC 20375	1
9.	Mr. Gary Harmon Silicon Compiler Systems Corporation 2045 Hamilton Ave. San Jose, California 95125	1
10.	Mr. Sam Nicolino Silicon Compiler Systems Corporation 2045 Hamilton Ave. San Jose, California 92152	1
11.	Dr. Allen Ross Consultant 6604 Lisa Lane Bowie, Maryland 20715	1

12. Prof. Donald E. Kirk 1
Associate Dean of Engineering
School of Engineering
San Jose State University
San Jose, California 95192

13. Mr. Howard Z. Bogert 1
Consultant
20720 4th Street, #12
Saratoga, California 95070

14. LT John C. Davidson 2
Department Head School, Class 108
Surface Warfare Officers School Command
Newport, Rhode Island 02841-5012

15. Dr. Waldo G. Magnuson 1
Lawrence Livermore National Laboratory
L-156
PO Box 808
Livermore, California 94550

Thesis
D1644
c.1

Davidson
Implementation of a
Design for Testability
strategy using the
Genesil silicon complier.